Bulletin of the Technical Committee on

Data Engineering

March 2009 Vol. 32 No. 1

IEEE Computer Society

Letters

Letter from the Editor-in-Chief . . . . *David Lomet* 1
Letter from the Guest Editors . . . . *Beng Chin Ooi and Srinivasan Parthasarathy* 2

Special Issue on Data Management on Cloud Computing Platforms

Data Management in the Cloud: Limitations and Opportunities . . . . *Daniel J. Abadi* 3
Deploying Database Appliances in the Cloud . . . .
Ashraf Aboulnaga, Kenneth Salem, Ahmed A. Soror, Umar Farooq Minhas, Peter Kokosielis and Sunil Kamath 13
Privacy-preserving Digital Identity Management for Cloud Computing . . . .
. . . . *Elisa Bertino, Federica Paci, Rodolfo Ferrini and Ning Shang* 21
Towards a Scalable Enterprise Content Analytics Platform . . . .
. . . . . . *Kevin Beyer, Vuk Ercegovac, Rajasekar Krishnamurthy, Sriram Raghavan, Jun Rao, Frederick Reiss, Eugene J. Shekita, David Simmen, Sandeep Tata, Shivakumar Vaithyanathan and Huaiyu Zhu* 28
Building a Cloud for Yahoo! . . . .
. . . . . *Brian F. Cooper, Eric Baldeschwieler, Rodrigo Fonseca, James J. Kistler, P.P.S. Narayan, Chuck Neerdaels, Toby Negrin, Raghu Ramakrishnan, Adam Silberstein, Utkarsh Srivastava and Raymie Stata* 36
On the Varieties of Clouds for Data Intensive Computing . . . . . . *Robert L. Grossman and Yunhong Gu* 44
Optimizing Utility in Cloud Computing through Autonomic Workload Execution . . . .
. . . . . *Norman W. Paton, Marcelo A. T. de Aragão, Kevin Lee, Alvaro A. A. Fernandes and Rizos Sakellariou* 51
Implementation Issues of A Cloud Computing Platform . . . . *Bo Peng, Bin Cui and Xiaoming Li* 59
Dataflow Processing and Optimization on Grid and Cloud Infrastructures . . . . *M. Tsangaris, G. Kakaletris, H. Kllapi, G. Papanikos, F. Pentaris, P. Polydoras, E. Sitaridi, V. Stoumpos and Y. Ioannidis* 67
An Indexing Framework for Efficient Retrieval on the Cloud . . . . *Sai Wu and Kun-Lung Wu* 75

Conference and Journal Notices

ICDE 2009 Conference . . . . back cover

The TC on Data Engineering
Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TC on Data Engineering web page is
http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin
The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at
http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

International Conference on Data Engineering

ICDE (the International Conference on Data Engineering) is the flagship database conference of the IEEE. The 2009 ICDE will be held in Shanghai, China at the end of March. I would encourage readers to check the ”Call for Participation” on the back inside cover of this issue of the Bulletin for more details. ICDE has become not only one of the best database conferences, but one of the largest as well. I attend this conference every year and always find my time well spent. Not only is the research program first-rate, but there is an industrial program, demos, and workshops as well.

The Current Issue

This issue revisits “cloud” based data management. An earlier Bulletin issue (December, 2006) gave a preliminary look at this general area, outlining some of the promise and opportunity, but largely before there was much in the way of real data management in the cloud. That is no longer the case. The past few years have seen an explosion of interest and work in this area. But while we now have some experience and insight as well, this is an area that will continue to be a challenge and opportunity for a long time.

Why a long time? This is a combination of: (1) the cloud is a platform that will be of increasing importance over a very long period, as our industry makes a phase change from in-house data management to cloud-hosted data management; (2) the problems associated with dealing with cloud data are formidable, from performance issues to security and privacy, from metadata management to high availability. These aspects to issues make them great research areas– importance and challenge. Providing solutions will help to enable the dream not only of ”information at your fingertips”, but also ”available wherever you happen to be”.

Beng Chin Ooi and Srinivasan Parthasarathy have assembled an issue that contains a very interesting cross section of the work that is going on right now, both at academic institutions and in industry. There are consortia that have emerged that make it possible for researchers, regardless of where they are, to join in the “cloud” effort. So I hope these papers provide a way to entice some of you to enroll in this effort. Beng Chin and Srinivasan have done a fine job assembling this issue, very successfully involving both academia and industry. I am sure you will find it of great interest.

David Lomet
Microsoft Corporation

Letter from the Guest Editors

This special issue brings to bear recent advances in the field of cloud computing that are applicable to data management, data retrieval, data intensive applications and data analysis applications.

Cloud computing represents an important step towards realizing McCarthy's dream that all aspects of computation may some day be organized as a public utility service. It embraces concepts such as *software as a service* and *platform as a service*, which incorporate services for workflow facilities for application design and development, deployment and hosting services, data integration, and network-based access to and management of software. Customers of clouds, much like customers of utility companies, can subscribe to different services at different service levels to guarantee the desired quality of service.

Search and electronic commerce companies such as Google, Microsoft, Amazon and Yahoo have adopted cloud computing technology in a major way as have Fortune 500 companies such as IBM, General Electric and Procter and Gamble. Academic units and government agencies are also key players, and several joint efforts among them are at the forefront of cloud computing technology.

In this special issue, we hope to offer readers a brief glimpse into this exciting new technology, specifically from the perspective of data management and data intensive applications. The articles in this issue cover a wide array of issues on topics that range from specific instantiations of cloud computing technology to the broader perspective of the opportunities and possibilities that the technology open up.

Abadi reviews and discusses the limitations of the cloud computing paradigm, and offers a perspective on the potential opportunities for data management and analysis applications within the paradigm.

Aboulnaga *et al.* discuss the challenges in deploying database appliances on Infrastructure as a Service clouds, as well as the tools and techniques for addressing the issues.

Bertino *et al.* highlight the important issue of privacy preservation in cloud computing systems and describe an approach to privacy preservation in such systems while at the same time enhancing interoperability across domains and simplifying existing identity verification policies.

Beyer *et al.* describe the key features of an enterprise content analysis platform being developed at IBM Almaden which targets the analysis of semi-structured content.

Cooper *et al.* highlight the significant challenges with building a commercial cloud computing system at Yahoo that emphasizes data storage, processing and querying capabilities.

Grossman and Gu provide a nice overview of cloud computing and what differentiates it from past work. They also distinguish among different types of cloud technology in existence today, and conclude with a discussion on open research problems in the arena.

Paton *et al.* describe an autonomic utility-based approach to adaptive workload execution (scheduling), and illustrate the benefits of their approach on workloads comprising workflows and queries.

Peng, Cui and Li discuss lessons learned from constructing a cloud computing platform in an academic environment, and discuss potential improvements to facilitate massive data processing and enhanced system throughput in the context of a specific web and text mining application domain.

Tsangaris *et al.* describe an ongoing system project called Athena Distributed Processing (ADP), its key components and its challenges within the context of supporting user defined operators and enabling efficient dataflow processing and optimization on grid and cloud infrastructures.

Wu and Wu propose a new indexing framework for cloud computing systems based on the Peer-to-Peer structured overlay network concept that supports efficient dynamic network expansion and shrinkage, and demonstrate its viability on the Amazon EC2 cloud.

We would like to thank Shirish Tatikonda and Sai Wu for their help in assembling this issue. We hope you enjoy reading it.

Beng Chin Ooi and Srinivasan Parthasarathy
National University of Singapore and Ohio State University

Data Management in the Cloud: Limitations and Opportunities

Daniel J. Abadi
Yale University
New Haven, CT, USA
dna@cs.yale.edu

Abstract

Recently the cloud computing paradigm has been receiving significant excitement and attention in the media and blogosphere. To some, cloud computing seems to be little more than a marketing umbrella, encompassing topics such as distributed computing, grid computing, utility computing, and software-as-a-service, that have already received significant research focus and commercial implementation. Nonetheless, there exist an increasing number of large companies that are offering cloud computing infrastructure products and services that do not entirely resemble the visions of these individual component topics.

In this article we discuss the limitations and opportunities of deploying data management issues on these emerging cloud computing platforms (e.g., Amazon Web Services). We speculate that large scale data analysis tasks, decision support systems, and application specific data marts are more likely to take advantage of cloud computing platforms than operational, transactional database systems (at least initially). We present a list of features that a DBMS designed for large scale data analysis tasks running on an Amazon-style offering should contain. We then discuss some currently available open source and commercial database options that can be used to perform such analysis tasks, and conclude that none of these options, as presently architected, match the requisite features. We thus express the need for a new DBMS, designed specifically for cloud computing environments.

1 Introduction

Though not everyone agrees on the exact definition of cloud computing [32], most agree the vision encompasses a general shift of computer processing, storage, and software delivery away from the desktop and local servers, across the network, and into next generation data centers hosted by large infrastructure companies such as Amazon, Google, Yahoo, Microsoft, or Sun. Just as the electric grid revolutionized access to electricity one hundred years ago, freeing corporations from having to generate their own power, and enabling them to focus on their business differentiators, cloud computing is hailed as revolutionizing IT, freeing corporations from large IT capital investments, and enabling them to plug into extremely powerful computing resources over the network.

Data management applications are potential candidates for deployment in the cloud. This is because an on-premises enterprise database system typically comes with a large, sometimes prohibitive up-front cost, both in

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

hardware and in software. For many companies (especially for start-ups and medium-sized businesses), the pay-as-you-go cloud computing model, along with having someone else worrying about maintaining the hardware, is very attractive. In this way, cloud computing is reminiscent of the application service provider (ASP) and database-as-a-service (DaaS) paradigms. In practice, cloud computing platforms, like those offered by Amazon Web Services, AT&T's Synaptic Hosting, AppNexus, GoGrid, Rackspace Cloud Hosting, and to an extent, the HP/Yahoo/Intel Cloud Computing Testbed, and the IBM/Google cloud initiative, work differently than ASPs and DaaS. Instead of owning, installing, and maintaining the database software for you (often in a multi-tenancy architecture), cloud computing vendors typically maintain little more than the hardware, and give customers a set of virtual machines in which to install their own software. Resource availability is typically elastic, with a seemingly infinite amount compute power and storage available on demand, in a pay-only-for-what-you-use pricing model.

This article explores the advantages and disadvantages of deploying database systems in the cloud. We look at how the typical properties of commercially available cloud computing platforms affect the choice of data management applications to deploy in the cloud. Due to the ever-increasing need for more analysis over more data in today's corporate world, along with an architectural match in currently available deployment options, we conclude that read-mostly analytical data management applications are better suited for deployment in the cloud than transactional data management applications. We thus outline a research agenda for large scale data analysis in the cloud, showing why currently available systems are not ideally-suited for cloud deployment, and arguing that there is a need for a newly designed DBMS, architected specifically for cloud computing platforms.

2 Data Management in the Cloud

Before discussing in Section 3 the features a database system must implement for it to run well in the cloud, in this section we attempt to narrow the scope of potential database applications to consider for cloud deployment. Our goal in this section is to decide which data management applications are best suited for deployment on top of cloud computing infrastructure. In order to do this, we first discuss three characteristics of a cloud computing environment that are most pertinent to the ensuing discussion.

2.1 Cloud Characteristics

Compute power is elastic, but only if workload is parallelizable. One of the oft-cited advantages of cloud computing is its elasticity in the face of changing conditions. For example, during seasonal or unexpected spikes in demand for a product retailed by an e-commerce company, or during an exponential growth phase for a social networking Website, additional computational resources can be allocated on the fly to handle the increased demand in mere minutes (instead of the many days it can take to procure the space and capital equipment needed to expand the computational resources in-house). Similarly, in this environment, one only pays for what one needs, so increased resources can be obtained to handle spikes in load and then released once the spike has subsided. However, getting additional computational resources is not as simple as a magic upgrade to a bigger, more powerful machine on the fly (with commensurate increases in CPUs, memory, and local storage); rather, the additional resources are typically obtained by allocating additional server instances to a task. For example, Amazon's Elastic Compute Cloud (EC2) apportions computing resources in small, large, and extra large virtual private server instances, the largest of which contains no more than four cores. If an application is unable to take advantage of the additional server instances by offloading some of its required work to the new instances which run in parallel with the old instances, then having the additional server instances available will not be much help. In general, applications designed to run on top of a shared-nothing architecture (where a set of independent machines accomplish a task with minimal resource overlap) are well suited for such an environment. Some cloud computing products, such as Google's App Engine, provide not only a cloud computing infrastructure, but also a complete software stack with a restricted API so that software developers are forced to write programs that can run in a shared-nothing environment and thus facilitate elastic scaling.

Data is stored at an untrusted host. Although it may not seem to make business sense for a cloud computing host company to violate the privacy of its customers and access data without permission, such a possibility nevertheless makes some potential customers nervous. In general, moving data off premises increases the number of potential security risks, and appropriate precautions must be made. Furthermore, although the name "cloud computing" gives the impression that the computing and storage resources are being delivered from a celestial location, the fact is, of course, that the data is physically located in a particular country and is subject to local rules and regulations. For example, in the United States, the US Patriot Act allows the government to demand access to the data stored on any computer; if the data is being hosted by a third party, the data is to be handed over without the knowledge or permission of the company or person using the hosting service [1]. Since most cloud computing vendors give the customer little control over where data is stored (e.g., Amazon S3 only allows a customer to choose between US and EU data storage options), the customer has little choice but to assume the worst and that unless the data is encrypted using a key not located at the host, the data may be accessed by a third party without the customer's knowledge.

Data is replicated, often across large geographic distances Data availability and durability is paramount for cloud storage providers, as data loss or unavailability can be damaging both to the bottom line (by failing to hit targets set in service level agreements [2]) and to business reputation (outages often make the news [3]). Data availability and durability are typically achieved through under-the-covers replication (i.e., data is automatically replicated without customer interference or requests). Large cloud computing providers with data centers spread throughout the world have the ability to provide high levels of fault tolerance by replicating data across large geographic distances. Amazon's S3 cloud storage service replicates data across "regions" and "availability zones" so that data and applications can persist even in the face of failures of an entire location. The customer should be careful to understand the details of the replication scheme however; for example, Amazon's EBS (elastic block store) will only replicate data within the same availability zone and is thus more prone to failures.

2.2 Data management applications in the cloud

The above described cloud characteristics have clear consequences on the choice of what data management applications to move into the cloud. In this section we describe the suitability of moving the two largest components of the data management market into the cloud: transactional data management and analytical data management.

2.2.1 Transactional data management

By "transactional data management", we refer to the bread-and-butter of the database industry, databases that back banking, airline reservation, online e-commerce, and supply chain management applications. These applications typically rely on the ACID guarantees that databases provide, and tend to be fairly write-intensive. We speculate that transactional data management applications are *not* likely to be deployed in the cloud, at least in the near future, for the following reasons:

Transactional data management systems do not typically use a shared-nothing architecture. The transactional database market is dominated by Oracle, IBM DB2, Microsoft SQL Server, and Sybase [29]. Of these four products, neither Microsoft SQL Server nor Sybase can be deployed using a shared-nothing architecture. IBM released a shared-nothing implementation of DB2 in the mid-1990s which is now available as a "Database Partitioning Feature" (DPF) add-on to their flagship product [4], but is designed to help scale analytical applications running on data warehouses, not transactional data management [5]. Oracle had no shared-nothing implementation until very recently (September 2008 with the release of the Oracle Database Machine that uses a shared-nothing architecture at the storage layer), but again, this implementation is designed only to be used for data warehouses [6].

Implementing a transactional database system using a shared-nothing architecture is non-trivial, since data is partitioned across sites and, in general, transactions can not be restricted to accessing data from a single site. This results in complex distributed locking and commit protocols, and in data being shipped over the network leading to increased latency and potential network bandwidth bottlenecks. Furthermore the main benefit of a shared-nothing architecture is its scalability [24]; however this advantage is less relevant for transactional data processing for which the overwhelming majority of deployments are less than 1 TB in size [33].

It is hard to maintain ACID guarantees in the face of data replication over large geographic distances. The CAP theorem [19] shows that a shared-data system can only choose at most two out of three properties: consistency, availability, and tolerance to partitions. When data is replicated over a wide area, this essentially leaves just consistency and availability for a system to choose between. Thus, the 'C' (consistency) part of ACID is typically compromised to yield reasonable system availability.

In order to get a sense of the inherent issues in building a replicated database over a wide area network, it is interesting to note the design approaches of some recent systems. Amazon's SimpleDB [11] and Yahoo's PNUTS [15] both implement shared-nothing databases over a wide-area network, but overcome the difficulties of distributed replication by relaxing the ACID guarantees of the system. In particular, they weaken the consistency model by implementing various forms of eventual/timeline consistency so that all replicas do not have to agree on the current value of a stored value (avoiding distributed commit protocols). Similarly, the research done by Brantner et. al. found that they needed to relax consistency and isolation guarantees in the database they built on top of Amazon's S3 storage layer [12]. Google's Bigtable [6] implements a replicated shared-nothing database, but does not offer a complete relational API and weakens the 'A' (atomicity) guarantee from ACID. In particular, it is a simple read/write store; general purpose transactions are not implemented (the only atomic actions are read-modify-write sequences on data stored under a single row key). SimpleDB and Microsoft SQL Server Data Services work similarly. The H-Store project [33] aims to build wide-area shared-nothing transactional database that adheres to strict ACID guarantees by using careful database design to minimize the number of transactions that access data from multiple partitions; however, the project remains in the vision stage, and the feasibility of the approach on a real-world dataset and query workload has yet to be demonstrated.

There are enormous risks in storing transactional data on an untrusted host. Transactional databases typically contain the complete set of operational data needed to power mission-critical business processes. This data includes detail at the lowest granularity, and often includes sensitive information such as customer data or credit card numbers. Any increase in potential security breaches or privacy violations is typically unacceptable.

We thus conclude that transactional data management applications are not well suited for cloud deployment. Despite this, there are a couple of companies that will sell you a transactional database that can run in Amazon's cloud: EnterpriseDB's Postgres Plus Advanced Server and Oracle. However, there has yet to be any published case studies of customers successfully implementing a mission critical transactional database using these cloud products and, at least in Oracle's case, the cloud version seems to be mainly intended for database backup [27].

2.2.2 Analytical data management

By "analytical data management", we refer to applications that query a data store for use in business planning, problem solving, and decision support. Historical data along with data from multiple operational databases are all typically involved in the analysis. Consequently, the scale of analytical data management systems is generally larger than transactional systems (whereas 1TB is large for transactional systems, analytical systems are increasingly crossing the petabyte barrier [25, 7]). Furthermore, analytical systems tend to be read-mostly (or read-only), with occasional batch inserts. Analytical data management consists of $3.98 billion [35] of the $14.6 billion database market [29] (27%) and is growing at a rate of 10.3% annually [35]. We speculate that analytical data management systems are well-suited to run in a cloud environment, and will be among the first data management applications to be deployed in the cloud, for the following reasons:

Shared-nothing architecture is a good match for analytical data management. Teradata, Netezza, Greenplum, DATAllegro (recently acquired by Microsoft), Vertica, and Aster Data all use a shared-nothing architecture (at least in the storage layer) in their analytical DBMS products, with IBM DB2 and recently Oracle also adding shared-nothing analytical products. The ever increasing amount of data involved in data analysis workloads is the primary driver behind the choice of a shared-nothing architecture, as the architecture is widely believed to scale the best [24]. Furthermore, data analysis workloads tend to consist of many large scan scans, multidimensional aggregations, and star schema joins, all of which are fairly easy to parallelize across nodes in a shared-nothing network. Finally, the infrequent writes in the workload eliminates the need for complex distributed locking and commit protocols.

ACID guarantees are typically not needed. The infrequent writes in analytical database workloads, along with the fact that it is usually sufficient to perform the analysis on a recent snapshot of the data (rather than on up-to-the-second most recent data) makes the 'A', 'C', and 'I' (atomicity, consistency, and isolation) of ACID easy to obtain. Hence the consistency tradeoffs that need to be made as a result of the distributed replication of data in transactional databases are not problematic for analytical databases.

Particularly sensitive data can often be left out of the analysis. In many cases, it is possible to identify the data that would be most damaging should it be accessed by a third party, and either leave it out of the analytical data store, include it only after applying an anonymization function, or include it only after encrypting it. Furthermore, less granular versions of the data can be analyzed instead of the lowest level, most detailed data.

We conclude that the characteristics of the data and workloads of typical analytical data management applications are well-suited for cloud deployment. The elastic compute and storage resource availability of the cloud is easily leveraged by a shared-nothing architecture, while the security risks can be somewhat alleviated. In particular, we expect the cloud to be a preferred deployment option for data warehouses for medium-sized businesses (especially those that do not currently have data warehouses due to the large up-front capital expenditures needed to get a data warehouse project off the ground), for sudden or short-term business intelligence projects that arise due to rapidly changing business conditions (e.g., a retail store analyzing purchasing patterns in the aftermath of a hurricane), and for customer-facing data marts that contain a window of data warehouse data intended to be viewed by the public (for which data security is not an issue).

3 Data Analysis in the Cloud

Now that we have settled on analytic database systems as a likely segment of the DBMS market to move into the cloud, we explore various currently available software solutions to perform the data analysis. We focus on two classes of software solutions: MapReduce-like software, and commercially available shared-nothing parallel databases. Before looking at these classes of solutions in detail, we first list some desired properties and features that these solutions should ideally have.

3.1 Cloud DBMS Wish List

Efficiency. Given that cloud computing pricing is structured in a way so that you pay for only what you use, the price increases linearly with the requisite storage, network bandwidth, and compute power. Hence, if data analysis software product A requires an order of magnitude more compute units than data analysis software product B to perform the same task, then product A will cost (approximately) an order of magnitude more than B. Efficient software has a direct effect on the bottom line.

Fault Tolerance. Fault tolerance in the context of analytical data workloads is measured differently than fault tolerance in the context of transactional workloads. For transactional workloads, a fault tolerant DBMS can recover from a failure without losing any data or updates from recently committed transactions, and in the context of distributed databases, can successfully commit transactions and make progress on a workload even in the face of worker node failure. For read-only queries in analytical workloads, there are no write transactions to commit, nor updates to lose upon node failure. Hence, a fault tolerant analytical DBMS is simply one that does not have to restart a query if one of the nodes involved in query processing fails. Given the large of amount of data that needs to be accessed for deep analytical queries, combined with the relatively weak compute capacity of a typical cloud compute server instance (e.g., a default compute unit on Amazon's EC2 service is the equivalent of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor), complex queries can involve hundreds (even thousands) of server instances and can take hours to complete. Furthermore, clouds are typically built on top of cheap, commodity hardware, for which failure is not uncommon. Consequently, the probability of a failure occurring during a long-running data analysis task is relatively high; Google, for example, reports an average of 1.2 failures per analysis job [7]. If a query must restart each time a node fails, then long, complex queries are difficult to complete.

Ability to run in a heterogeneous environment. The performance of cloud compute nodes is often not consistent, with some nodes attaining orders of magnitude worse performance than other nodes. There are a variety of reasons why this could occur, ranging from hardware failure causing degraded performance on a node [31], to an instance being unable to access the second core on a dual-core machine [8], to contention for non-virtualized resources. If the amount of work needed to execute a query is equally divided amongst the cloud compute nodes, then there is a danger that the time to complete the query will be approximately equal to time for the slowest compute node to complete its assigned task. A node observing degraded performance would thus have a disproportionate affect on total query latency. A system designed to run in a heterogeneous environment would take appropriate measures to prevent this from occurring.

Ability to operate on encrypted data. As mentioned in Section 2.2.2, sensitive data may be encrypted before being uploaded to the cloud. In order to prevent unauthorized access to the sensitive data, any application running in the cloud should not have the ability to directly decrypt the data before accessing it. However, shipping entire tables or columns out of the cloud for decryption is bandwidth intensive. Hence, the ability of the data analysis system to operate directly on encrypted data (such as in [10, 20, 18, 23, 28]) so that a smaller amount of data needs to ultimately be shipped elsewhere to be decrypted could significantly improve performance.

Ability to interface with business intelligence products. There are a variety of customer-facing business intelligence tools that work with database software and aid in the visualization, query generation, result dashboarding, and advanced data analysis. These tools are an important part of the analytical data management picture since business analysts are often not technically advanced and do not feel comfortable interfacing with the database software directly. These tools typically interface with the database using ODBC or JDBC, so database software that want to work these products must accept SQL queries over these connections.

Using these desired properties of our cloud data analysis software, we now examine how close two currently available solutions come to attaining these properties: MapReduce-like software, and commercially available shared-nothing parallel databases.

3.2 MapReduce-like software

MapReduce [7] and related software such as the open source Hadoop [9], useful extensions [30], and Microsoft's Dryad/SCOPE stack [13] are all designed to automate the parallelization of large scale data analysis workloads. Although DeWitt and Stonebraker took a lot of criticism for comparing MapReduce to database systems in their recent controversial blog posting [17] (many believe that such a comparison is apples-to-oranges), a comparison is warranted since MapReduce (and its derivatives) is in fact a useful tool for performing data analysis in the cloud [9]. The MapReduce programming model and framework implementation satisfies many of the previously stated desired properties:

Fault Tolerance. MapReduce is designed with fault tolerance as a high priority. A data analysis job is divided into many small tasks and upon a failure, tasks assigned to a failed machine are transparently reassigned to another machine. Care is taken to make sure that partially executed tasks are not doubly accounted for in the final query result. In a set of experiments in the original MapReduce paper, it was shown that explicitly killing 200 out of 1746 worker processes involved in a MapReduce job resulted in only a 5% degradation in query performance [7].

Ability to run in a heterogeneous environment. MapReduce is also carefully designed to run in a heterogeneous environment. Towards the end of a MapReduce job, tasks that are still in progress get redundantly executed on other machines, and a task is marked as completed as soon as either the primary or the backup execution has completed. This limits the effect that "straggler" machines can have on total query time, as backup executions of the tasks assigned to these machines will complete first. In a set of experiments in the original MapReduce paper, it was shown that backup task execution improves query performance by 44% by alleviating the adverse affect caused by slower machines.

Ability to operate on encrypted data. Neither MapReduce, nor its derivatives, come with a native ability to operate on encrypted data. Such an ability would have to be provided using user-defined code.

Ability to interface with business intelligence products. Since MapReduce is not intended to be a database system, it is not SQL compliant and thus it does not easily interface with existing business intelligence products.

Efficiency. The efficiency and raw performance of MapReduce is a matter of debate. A close inspection of the experimental results presented in the MapReduce paper [7] would seem to indicate that there is room for performance improvement. Figure 2 of the paper shows the performance of a simple grep query where a rare string is searched for inside a 1TB dataset. In this query, 1TB of data is read off of the 3600 disks in the cluster (in parallel) and a very simple pattern search is performed. Disk should clearly be the bottleneck resource since the string is rare, so query results do not need to be shipped over the network, and the query is computationally trivial. Despite these observations, the entire grep query takes 150 seconds to complete. If one divides the 1TB of data by the 3600 disks and 150 seconds to run the query, the average throughput with which data is being read is less than 2 MB/s/disk. At peak performance, MapReduce was reading data at 32GB/s which is less than 10MB/s/disk. Given the long start-up time to get to peak performance, and the fact that peak performance is four to six times slower than how fast disks in the cluster could actually be read, there indeed is room for improvement. Other benchmarks [36] (albeit not performed up to the standards of publishable academic rigor) have also shown MapReduce to be about an order of magnitude slower than alternative systems.

Much of the performance issues of MapReduce and its derivative systems can be attributed to the fact that they were not initially designed to be used as complete, end-to-end data analysis systems over structured data. Their target use cases include scanning through a large set of documents produced from a web crawler and producing a web index over them [7]. In these applications, the input data is often unstructured and a brute force scan strategy over all of the data is usually optimal. MapReduce then helps automate the parallelization of the data scanning and application of user defined functions as the data is being scanned.

For more traditional data analysis workloads of the type discussed in Section 2.2.2 that work with data produced from business operational data stores, the data is far more structured. Furthermore, the queries tend to access only a subset of this data (e.g., breakdown the profits of stores *located in the Northeast*). Using data structures that help accelerate access to needed entities (such as indexes) and dimensions (such as column-stores), and data structures that precalculate common requests (such as materialized views) often outperform a brute-force scan execution strategy.

Many argue that the lack of these "helper" data structures in MapReduce is a feature, not a limitation. These additional structures generally require the data to be loaded into to data analysis system before it can be used. This means that someone needs to spend time thinking about what schema to use for the data, define the schema and load the data into it, and decide what helper data structures to create (of course self-managing/self-tuning systems can somewhat alleviate this burden). In contrast, MapReduce can immediately read data off of the file system and answer queries on-the-fly without any kind of loading stage.

Nonetheless, at the complexity cost of adding a loading stage, indexes, columns, and materialized views unquestionably can improve performance of many types of queries. If these data structures are utilized to improve the performance of multiple queries, then the one-time cost of their creation is easily outweighed by the benefit each time they are used.

The absence of a loading phase into MapReduce has additional performance implications beyond precluding the use of helper data structures. During data load, data can be compressed on disk. This can improve performance, even for brute-force scans, by reducing the I/O time for subsequent data accesses. Furthermore, since data is not loaded in advance, MapReduce needs to perform data parsing at runtime (using user-defined code) each time the data is accessed, instead of parsing the data just once at load time.

The bottom line is that the performance of MapReduce is dependent on the applications that it is used for. For complex analysis of unstructured data (which MapReduce was initially designed for) where brute-force scans is the right execution strategy, MapReduce is likely a good fit. But for the multi-billion dollar business-oriented data analysis market, MapReduce can be wildly inefficient.

3.3 Shared-Nothing Parallel Databases

A more obvious fit for data analysis in the cloud are the commercially available shared-nothing parallel databases, such as Teradata, Netezza, IBM DB2, Greenplum, DATAllegro, Vertica, and Aster Data, that already hold a reasonable market share for on-premises large scale data analysis [35]. DB2, Greenplum, Vertica, and Aster Data are perhaps the most natural fit since they sell software-only products that could theoretically run in the data centers hosted by cloud computing providers. Vertica already markets a version of its product designed to run in Amazon's cloud [34].

Parallel databases implement a largely complimentary set of properties from our wish list relative to MapReduce-like software:

Ability to interface with business intelligence products. Given that the business intelligence products are designed to work on top of databases, this property essentially comes for free. More mature databases, such as DB2, tend to have carefully optimized and certified interfaces with a multitude of BI products.

Efficiency At the cost of the additional complexity in the loading phase discussed in Section 3.2, parallel databases implement indexes, materialized views, and compression to improve query performance.

Fault Tolerance. Most parallel database systems restart a query upon a failure. This is because they are generally designed for environments where queries take no more than a few hours and run on no more than a few hundred machines. Failures are relatively rare in such an environment, so an occasional query restart is not problematic. In contrast, in a cloud computing environment, where machines tend to be cheaper, less reliable, less powerful, and more numerous, failures are more common. Not all parallel databases, however, restart a query upon a failure; Aster Data reportedly has a demo showing a query continuing to make progress as worker nodes involved in the query are killed [26].

Ability to run in a heterogeneous environment. Parallel databases are generally designed to run on homogeneous equipment and are susceptible to significantly degraded performance if a small subset of nodes in the parallel cluster are performing particularly poorly.

Ability to operate on encrypted data. Commercially available parallel databases have not caught up to (and do not implement) the recent research results on operating directly on encrypted data. In some cases simple operations (such as moving or copying encrypted data) are supported, but advanced operations, such as performing aggregations on encrypted data, is not directly supported. It should be noted, however, that it is possible to hand-code encryption support using user defined functions.

3.4 A Call For A Hybrid Solution

It is now clear that neither MapReduce-like software, nor parallel databases are ideal solutions for data analysis in the cloud. While neither option satisfactorily meets all five of our desired properties, each property (except the primitive ability to operate on encrypted data) is met by at least one of the two options. Hence, a hybrid solution that combines the fault tolerance, heterogeneous cluster, and ease of use out-of-the-box capabilities of MapReduce with the efficiency, performance, and tool plugability of shared-nothing parallel database systems could have a significant impact on the cloud database market.

There has been some recent work on bringing together ideas from MapReduce and database systems, however, this work focuses mainly on language and interface issues. The Pig project at Yahoo [30] and the SCOPE project at Microsoft [13] aim to integrate declarative query constructs from the database community into MapReduce-like software to allow greater data independence, code reusability, and automatic query optimization. Greenplum and Aster Data have added the ability to write MapReduce functions (instead of, or in addition to, SQL) over data stored in their parallel database products [22]. Although these four projects are without question an important step in the direction of a hybrid solution, there remains a need for a hybrid solution at the systems level in addition to at the language level.

One interesting research question that would stem from such a hybrid integration project would be how to combine the ease-of-use out-of-the-box advantages of MapReduce-like software with the efficiency and shared-work advantages that come with loading data and creating performance enhancing data structures. Incremental algorithms are called for, where data can initially be read directly off of the file system out-of-the-box, but each time data is accessed, progress is made towards the many activities surrounding a DBMS load (compression, index and materialized view creation, etc.).

Another interesting research question is how to balance the tradeoffs between fault tolerance and performance. Maximizing fault tolerance typically means carefully checkpointing intermediate results, but this usually comes at a performance cost (e.g., the rate which data can be read off disk in the sort benchmark from the original MapReduce paper is half of full capacity since the same disks are being used to write out intermediate Map output). A system that can adjust its levels of fault tolerance on the fly given an observed failure rate could be one way to handle the tradeoff.

The bottom line is that there is both interesting research and engineering work to be done in creating a hybrid MapReduce/parallel database system.

4 Acknowledgments

We thank Andy Pavlo and Samuel Madden for their feedback on this article. The author is funded by the NSF under grants IIS-0845643 and IIS-0844480.

References

[1] http://news.bbc.co.uk/1/hi/technology/7421099.stm.
[2] http://aws.amazon.com/s3-sla/.
[3] http://wiki.cloudcommunity.org/wiki/CloudComputing:Incidents_Database.
[4] http://en.wikipedia.org/wiki/IBM_DB2.
[5] http://www.ibm.com/developerworks/db2/library/techarticle/dm-0608mcinerney/index.html.

[6] http://www.oracle.com/solutions/business_intelligence/exadata.html.

[7] http://www.sybase.com/detail?id=1054047.

[8] http://developer.amazonwebservices.com/connect/thread.jspa?threadID=16912.

[9] http://www.lexemetech.com/2008/08/elastic-hadoop-clusters-with-amazons.html.

[10] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proc. of SIGMOD*, pages 563–574, 2004.

[11] Amazon Web Services. SimpleDB. Web Page. http://aws.amazon.com/simpledb/.

[12] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska. Building a Database on S3. In *Proc. of SIGMOD*, pages 251–264, 2008.

[13] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and efficient parallel processing of massive data sets. In *Proc. of VLDB*, 2008.

[14] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of OSDI*, 2006.

[15] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!s hosted data serving platform. In *Proceedings of VLDB*, 2008.

[16] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. pages 137–150, December 2004.

[17] D. DeWitt and M. Stonebraker. MapReduce: A major step backwards. DatabaseColumn Blog. http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html.

[18] T. Ge and S. Zdonik. Answering aggregation queries in a secure system model. In *Proc. of VLDB*, pages 519–530, 2007.

[19] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[20] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proc. of SIGMOD*, pages 216–227, 2002.

[21] Hadoop Project. Welcome to Hadoop! Web Page. http://hadoop.apache.org/core/.

[22] J. N. Hoover. Start-Ups Bring Google's Parallel Processing To Data Warehousing. InformationWeek, August 29th, 2008.

[23] M. Kantarcoglu and C. Clifton. Security issues in querying encrypted data. In *19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, 2004.

[24] S. Madden, D. DeWitt, and M. Stonebraker. Database parallelism choices greatly impact scalability. DatabaseColumn Blog. http://www.databasecolumn.com/2007/10/database-parallelism-choices.html.

[25] C. Monash. The 1-petabyte barrier is crumbling. http://www.networkworld.com/community/node/31439.

[26] C. Monash. Introduction to Aster Data and nCluster. DBMS2 Blog. http://www.dbms2.com/2008/09/02/introduction-to-aster-data-and-ncluster/.

[27] C. Monash. Oracle Announces an Amazon Cloud Offering. DBMS2 Blog. http://www.dbms2.com/2008/09/22/oracle-announces-an-amazon-cloud-offering/.

[28] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In *IFIP WG 11.3 on Data and Application Security*, 2006.

[29] C. Olofson. Worldwide RDBMS 2005 vendor shares. Technical Report 201692, IDC, May 2006.

[30] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD Conference*, pages 1099–1110, 2008.

[31] RightScale. Top reasons amazon ec2 instances disappear. http://blog.rightscale.com/2008/02/02/top-reasons-amazon-ec2-instances-disappear/.

[32] Slashdot. Multiple Experts Try Defining Cloud Computing. http://tech.slashdot.org/article.pl?sid=08/07/17/2117221.

[33] M. Stonebraker, S. R. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era (it's time for a complete rewrite). In *VLDB*, Vienna, Austria, 2007.

[34] Vertica. Performance On-Demand with Vertica Analytic Database for the Cloud. http://www.vertica.com/cloud.

[35] D. Vesset. Worldwide data warehousing tools 2005 vendor shares. Technical Report 203229, IDC, August 2006.

[36] E. Yoon. Hadoop Map/Reduce Data Processing Benchmarks. Hadoop Wiki. http://wiki.apache.org/hadoop/DataProcessingBenchmarks.

Deploying Database Appliances in the Cloud

Ashraf Aboulnaga* Kenneth Salem* Ahmed A. Soror* Umar Farooq Minhas*
Peter Kokosielis† Sunil Kamath†

*University of Waterloo
†IBM Toronto Lab

Abstract

Cloud computing is an increasingly popular paradigm for accessing computing resources. A popular class of computing clouds is Infrastructure as a Service (IaaS) clouds, exemplified by Amazon's Elastic Computing Cloud (EC2). In these clouds, users are given access to virtual machines on which they can install and run arbitrary software, including database systems. Users can also deploy database appliances on these clouds, which are virtual machines with pre-installed pre-configured database systems. Deploying database appliances on IaaS clouds and performance tuning and optimization in this environment introduce some interesting research challenges. In this paper, we present some of these challenges, and we outline the tools and techniques required to address them. We present an end-to-end solution to one tuning problem in this environment, namely partitioning the CPU capacity of a physical machine among multiple database appliances running on this machine. We also outline possible future research directions in this area.

1 Introduction

Cloud computing has emerged as a powerful and cost-effective paradigm for provisioning computing power to users. In the cloud computing paradigm, users use an intranet or the Internet to access a shared computing cloud that consists of a large number (thousands or tens of thousands) of interconnected machines organized as one or more clusters. This provides significant benefits both to providers of computing power and to users of this computing power. For providers of computing power, the push to cloud computing is driven by economies of scale. By operating massive clusters in specially designed and carefully located data centers, providers can reduce administrative and operating costs, such as the costs of power and cooling [15, 16]. In addition, the per-unit costs of hardware, software and networking become significantly cheaper at this scale [4]. For users, cloud computing offers simple and flexible resource provisioning without up-front equipment and set up costs and on-going administrative and maintenance burdens. Users can run software in the cloud, and they can grow and shrink the computing power available to this software in response to growing and shrinking load [4].

There are different flavors of cloud computing, depending on how much flexibility the user has to customize the software running in the cloud. In this paper, we focus on computing clouds where the user sees a bare-bones machine with just an operating system and gets full flexibility in installing and configuring software on this machine. These clouds are known as *Infrastructure as a Service (IaaS)* clouds. A very prominent example

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

of this type of cloud is Amazon's Elastic Computing Cloud (EC2) [2], which enables users to rent computing power from Amazon to run their software. Other providers of this style of cloud computing include GoGrid [13] and AppNexus [3]. Additionally, many organizations are building IaaS clouds for their internal use [6, 22].

In IaaS clouds, users are typically given access to *virtual machines (VMs)* [5, 23] on which they can install and run software. These virtual machines are created and managed by a *virtual machine monitor (VMM)* which is a layer of software between the operating system and the physical machine. The VMM controls the resources of the physical machine, and can create multiple VMs that share these physical machine resources. The VMs have independent operating systems running independent applications, and are isolated from each other by the VMM. The VMM controls the allocation of physical machine resources to the different VMs. The VMM also provides functionality such as saving and restoring the image of a running VM, or migrating VMs between physical machines.

A common model for deploying software in virtual machine environments is the *virtual appliance* model. A virtual appliance is a VM image with a pre-installed pre-configured application. Deploying the application simply requires copying this VM image to a physical machine, starting the VM, and performing any required configuration tasks. The cost of installing and configuring the application on the VM is incurred once, when the appliance is created, and does not need to be incurred again by users of the appliance. A *database appliance* is a virtual appliance where the installed application is a database system. With the increasing popularity of virtualization and cloud computing, we can expect that a common way of providing database services in the future will be through **database appliances deployed in IaaS clouds**. As an example of this deployment mode, Amazon offers MySQL, Oracle, and Microsoft SQL Server virtual appliances for deployment in its EC2 cloud.

An important question to ask is how to get the best database system performance in this environment. Cloud providers are interested in two related performance objectives: maximizing the utilization of cloud resources and minimizing the resources required to satisfy user demand. Users are interested in minimizing application response time or maximizing application throughput. Deploying database appliances in the cloud and tuning the database and virtualization parameters to optimize performance introduces some interesting research challenges. In this paper, we outline some of these challenges (Section 2), and we present the different tools and techniques required to address them (Section 3). We present our work on partitioning CPU capacity among database appliances as an example end-to-end tuning solution for virtualized environments (Section 4). We conclude by outlining some possible future research directions in this area (Section 5).

2 Deployment and Tuning Challenges

Our focus is on deploying and tuning virtual machines running database systems (i.e., database appliances) on large clusters of physical machines (i.e., computing clouds). This raises deployment and computing challenges, which we describe next.

2.1 Deployment Challenges

Creating a database appliance that can easily be deployed in a cloud, and obtaining an accessible, usable database instance from this appliance require addressing many issues related to deployment. These issues are not the research focus of our work, but we present them here since these seemingly simple and mundane tasks can be very tricky and time consuming. These issues include:

Localization:
When we start a VM from a copy of a database appliance, we need to give this new VM and the database system running on it a distinct "identity." We refer to this process as *localization*. For example, we need to give the VM a MAC address, an IP address, and a host name. We also need to adapt (or localize) the database instance running on this VM to the VM's new identity. For example, some database systems require every database

instance to have a unique name, which is sometimes based on the host name or IP address. The VMM and the underlying operating system and networking infrastructure may help with issues such as assigning IP addresses, but there is typically little support for localizing the database instance. The specific localization required varies from database system to database system, which increases the effort required for creating database appliances.

Routing:
In addition to giving every VM and database instance a distinct identity, we must be able to route application requests to the VM and database instance. This includes the IP-level routing of packets to the VM, but it also includes making sure that database requests are routed to the correct port and not blocked by any firewall, that the display is routed back to the client console if needed, that I/O requests are routed to the correct virtual storage device if the "compute" machines of the IaaS cloud are different from the storage machines, and so on.

Authentication:
The VM must be aware of the credentials of all clients that need to connect to it, independent of where it is run in the cloud.

2.2 Tuning Challenges

Next, we turn our attention to the challenges related to tuning the parameters of the virtualization environment and the database appliance to achieve the desired performance objectives. These are the primary focus of our research work, and they include:

Placement:
Virtualization allows the cloud provider to run a user's VM on any available physical machine. The mapping of virtual machines to physical machines can have a significant impact on performance. One simple problem is to decide how many virtual machines to run on each physical machine. The cloud provider would like to minimize the number of physical machines used, but running more VMs on a physical machine degrades the performance of these VMs. It is important to balance these conflicting objectives: minimizing the number of physical machines used while maintaining acceptable performance for users.

A more sophisticated mapping of virtual machines to physical machines could consider not only the number of VMs per physical machine, but also the resource requirements of these VMs. The placement algorithm could, for example, avoid mapping multiple I/O intensive VMs to the same physical machine to minimize I/O interference between these VMs. This type of mapping requires understanding the resource usage characteristics of the application running in the VM, which may be easier to do for database systems than for other types of applications since database systems have a highly stylized and often predictable resource usage pattern.

Resource Partitioning:
Another tuning challenge is to decide how to partition the resources of each physical machine among the virtual machines that are running on it. Most VMMs provide tools or APIs for controlling the way that physical resources are allocated. For example VMM scheduling parameters can be used to apportion the total physical CPU capacity among the VMs, or to control how virtual CPUs are mapped to physical CPUs. Other VMM parameters can be used to control the amount of physical memory that is available to each VM. To obtain the best performance, it is useful to take into account the characteristics of the application running in the VM so that we can allocate resources where they will provide the maximum benefit. Database systems can benefit from this application-informed resource partitioning, as we will show in Section 4.

Service Level Objectives:
To optimize the performance of a database appliance in a cloud environment, it is helpful to be able to express different *service level objectives*. The high-level tuning goal is to minimize the cloud resources required while maintaining adequate performance for the database appliance. Expressing this notion of "adequate performance" is not a trivial task. A database system is typically part of a multi-layer software stack that is used to

serve application requests. Service level agreements are typically expressed in terms of end-to-end application performance, with no indication of how much of this performance budget is available to the database system vs. how much is available to other layers of the software stack (e.g., the application server and the web server). Deriving the performance budget that is available to the database system for a given application request is not easy, since an application request can result in a varying number of database requests, and these database requests can vary greatly in complexity depending on the SQL statements being executed. Tuning in a cloud environment therefore requires developing practical and intuitive ways of expressing database service level objectives. Different workloads can have different service level objectives, and the tuning algorithms need to take these different service level objectives into account.

Dynamically Varying Workloads:
Tuning the performance of a database appliance (e.g., placement and resource partitioning) requires knowledge of the appliance's workload. The workload can simply be the full set of SQL statements that execute at the appliance. However, it is an interesting question whether there can be a more succinct but still useful representation of the workload. Another interesting question is whether some tuning decisions can be made without knowledge of the SQL statements (e.g., if this is a new database instance). It is also important to detect when the nature of the workload has changed, possibly by classifying the workload [11] and detecting when the workload class has changed. The tuning algorithms need to be able to deal with dynamically changing workloads that have different service level objectives.

3 Tools and Techniques

Next, we turn our attention to the tools and techniques that are needed to address the tuning challenges outlined above. These include:

Performance Models:
Predicting the effect of different tuning actions on the performance of a database appliance is an essential component of any tuning solution. This requires developing accurate and efficient performance models for database systems in virtualized environments. There are two general classes of models: white box models, which are based on internal knowledge of the database system, and black box models, which are typically statistical models based on external, empirical observations of the database system's performance.

White box modeling is especially attractive for database systems for two reasons. First, database systems have a stylized and constrained interface for user requests: they accept and execute SQL statements. This simplifies defining the inputs to the performance model. Second, and more importantly, database systems already have highly refined internal models of performance. One way to build a white box model is to expose these internal models to the tuning algorithm and adapt them to the tuning task at hand. For example, the query optimizer cost model, which has been used extensively as a what-if cost model for automatic physical database design [7], can be used to quantify the effect of allocating different shares of physical resources to a database appliance (see the next section for more details). Self-managing database systems have other internal models that can be exposed for use in performance tuning in a cloud environment. These include the memory consumption model used by a self-tuning memory manager [8, 21] or the model used for automatic diagnosis of performance problems [10].

The disadvantage of white box modeling is that the required performance models do not always exist in the database system, and developing white box models from scratch is difficult and time consuming. Even when internal models do exist in the database system, these models are sometimes not calibrated to accurately provide the required performance metric, and they sometimes make simplifying assumptions that ignore important aspects of the problem. For example, the query optimizer cost model is designed primarily to compare query execution plans, not to accurately estimate resource consumption. This cost model focuses on one query at a time, ignoring the sometimes significant effect of concurrently running interacting queries [1]. Because of these shortcomings of white box modeling, it is sometimes desirable to build black box models of performance by fitting statistical models to the observed results of performance experiments [1]. When building these models it is important to carefully decide which performance experiments to conduct to collect samples for the model, since these experiments can be costly and they have a considerable impact on model accuracy [18]. However, the illusion of infinite computing resources provided by IaaS clouds can actually simplify black box experimental modeling of database systems, since we can now easily provision as many machines as we need to run the performance experiments required for building an accurate model.

An interesting research question is whether it is possible to combine the best features of black box and white box modeling, by using the internal models of the database system as a starting point, but then refining these models based on experimental observations [12].

Optimization and Control Algorithms:
Solving the performance tuning problems of a cloud environment requires developing combinatorial optimization or automatic control algorithms that use the performance models described above to decide on the best tuning action. These algorithms can be static algorithms that assume a fixed workload, or they can be dynamic algorithms that adapt to changing workloads. The algorithms can simply have as a goal the best-effort maximization of performance [20], or they can aim to satisfy different service level objectives for different workloads [17].

Tools for System Administrators:
In addition to the models and algorithms described above, system administrators need tools for deploying and tuning database appliances. These tools should expose not only the performance characteristics of the VM, but also the performance characteristics of the database system running on this VM. For example, it would be useful to expose the what-if performance models of the database system to system administrators so that they can make informed tuning decisions, diagnose performance problems, or refine the recommendations of automatic tuning algorithms.

Co-tuning and Hint Passing:
The focus of the previous discussion has been on tuning virtual machine parameters. It is also important to tune the parameters of the database system running on this virtual machine. For example, if we decide to decrease the memory available to a VM running a database system, we need to decrease the sizes of the different memory pools of this database system. This *co-tuning* of VM and database system parameters is important to ensure that the tuning actions at one layer are coordinated with the tuning action at the other layer. Another way to coordinate VM tuning with database system tuning is to pass *hints* that can be used for tuning from the database system to the virtualization layer. These hints would contain information that is easy to obtain for the database system and useful for tuning at the virtualization layer. For example, these hints could be used to ensure that VM disks storing database objects (i.e., tables or indexes) that are accessed together are not mapped to the same physical disk. Information about which objects are accessed together is easily available to the database system and very useful to the virtualization layer.

4 Virtual Machine Configuration

In this section, we consider the following tuning problem: Given N virtual machines that share one physical machine, with each VM running an independent database system instance, how can we optimally partition the available CPU capacity of the physical machine among the virtual machines? Recall that the VMM provides mechanisms for deciding how much CPU capacity is allocated to each VM. We outline a solution to this resource partitioning problem below. Full details of our solution can be found in [19, 20].

We decide the partitioning of the available CPU capacity of the physical machine among the N virtual machines with the goal of maximizing the aggregate throughput of the N workloads (or minimizing their total completion time). This is a best-effort performance goal that does not consider explicit service level objectives for the different workloads.

The benefit that each database system will obtain from an increase in CPU allocation depends on that system's workload. We assume that we are given the set of SQL statements that make up the workload of each of the N database systems. These workloads represent the SQL statements executed by the different database systems in the same time interval, so the number of statements in a workload corresponds to its intensity (i.e., the rate of arrival of SQL statements). We assume that the workloads are fixed, and we do not deal with dynamically varying workloads.

To determine the best CPU partitioning, we need a model of the performance of a database workload as a function of the CPU capacity allocated to the VM running this workload. In our solution, we use the cost model of the database system's query optimizer as a what-if model to predict performance under different CPU allocations. This requires the query optimizer cost model to be aware of the effect of changing CPU capacity on performance. The cost model relies on one or more modeling parameters to describe CPU capacity and estimate the CPU cost of a query. We use different values of these CPU modeling parameters for different CPU allocations, thereby adding awareness of CPU allocation to the query optimizer cost model. We call such a cost model *virtualization aware*. The calibration procedure required to determine the values of the CPU modeling parameters to use for each CPU allocation is performed only once for every database system and physical machine configuration, and can be used for any workload that runs on this database system.

We use the virtualization aware cost models of the N database systems on the N VMs in a greedy search algorithm to determine the best partitioning of CPU capacity among the VMs. We also provide heuristics for refining the cost models based on comparing estimated performance to actual observed performance. We apply these refinement heuristics periodically, and we obtain a new partitioning of CPU capacity after each refinement of the cost model.

To illustrate the effectiveness of our approach, consider the following example (Figure 1). Using the Xen VMM [5] we created two virtual machines, each running an instance of PostgreSQL. We ran both VMs on the same physical machine, a Sun server with two 2.2GHz dual core AMD Opteron Model 275 x64 processors and 8GB memory, running SUSE Linux 10.1. For this example, we used a TPC-H database with scale factor 1. On one PostgreSQL instance we ran a workload consisting of three instances of TPC-H query $Q4$. On the other instance, we ran a workload consisting of nine instances of TPC-H query $Q13$. First, we allocated 50% of the available CPU capacity to each of the two virtual machines, ran the two workloads, and measured the total execution time of each workload. The results are illustrated by the bars on the left for each of the two workloads in Figure 1. Next, we repeated the experiment, but this time we allocated CPU capacity according to the recommendations of our CPU partitioning algorithm. The algorithm recommended giving 25% of the available CPU capacity to the first PostgreSQL instance (Workload 1) and the remaining 75% to the second instance (Workload 2). The execution times of the two workloads under this CPU allocation are shown in Figure 1 by the bars on the right for each of the two workloads. This change in CPU allocation reduces the execution time of the second workload by approximately 30%, while having little impact on the first workload. Thus, we can see the importance of correctly partitioning CPU capacity and the effectiveness of our approach to solving this problem.

5 Future Directions

The previous section illustrates a simple performance tuning problem in a cloud computing environment and its solution. Extending the research outlined in the previous section opens up many possibilities for future work, which we are exploring in our ongoing research activities. Instead of partitioning the resources of one physical machine among the VMs, we can consider *multiple* physical machines and partition their resources among the VMs, that is, decide which physical machine to use for each VM and what share of this machine's resources are

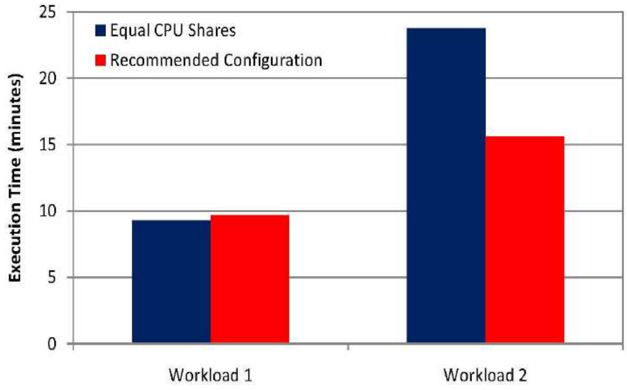


Figure 1: Effect of varying CPU allocation on workload performance.

allocated to the VM. We can also extend the work to deal with dynamically varying workloads, possibly with different explicit service level objectives. Another interesting research direction is improving the way we refine the query-optimizer-based cost model in response to observed performance.

Another interesting research direction is optimizing the allocation of I/O resources to different VMs. Some VMMs, such as VMWare ESX server [23], provide mechanisms for controlling how much of the I/O bandwidth of a physical machine is allocated to each VM running on this machine. Another mechanism to control the allocation of I/O resources to VMs is controlling the mapping of VM disks to physical disks. Using these two mechanisms to optimize the performance of database appliances is an interesting research direction, especially since many database workloads are I/O bound.

It would also be interesting to explore whether we can expose internal database system models other than the query optimizer cost model and use these models for tuning VM parameters or co-tuning VM and database system parameters. For example, the memory manager performance model can be used to control memory allocation.

The cloud environment also offers new opportunities, beyond the challenges of tuning database appliances. For example, since we can provision VMs on-demand, it would be interesting to explore the possibility of scaling out a database system to handle spikes in the workload by starting new replicas of this database system on newly provisioned VMs. This requires ensuring consistent access to the database during and after the replication process, coordinating request routing to the old and new VMs, and developing policies for when to provision and de-provision new replicas.

Finally, this idea of application-informed tuning of the virtualized environment is not restricted to database systems. This idea can be used for other types of applications that run in a cloud environment, such as large scale data analysis programs running on Map-Reduce style platforms [7, 9].

6 Conclusion

As cloud computing becomes more popular as a resource provisioning paradigm, we will increasingly see database systems being deployed as virtual appliances on Infrastructure as a Service (IaaS) clouds such as Amazon's EC2. In this paper, we outlined some of the challenges associated with deploying these appliances and tuning their performance, and we discussed the tools and techniques required to address these challenges. We presented an end-to-end solution to one tuning problem, namely partitioning the CPU capacity of a physical machine among the database appliances running on this machine. We also described some future directions for this research area. It is our belief that the style of application-informed tuning described in this paper can provide significant benefits to both providers and users of cloud computing.

References

[1] Mumtaz Ahmad, Ashraf Aboulnaga, Shivnath Babu, and Kamesh Munagala. Modeling and exploiting query interactions in database systems. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, 2008.

[2] Amazon EC2. http://aws.amazon.com/ec2/.

[3] AppNexus. http://www.appnexus.com/.

[4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley, Feb 2009.

[5] Paul T. Barham, Boris Dragovic, Keir Fraser, Steven Hand, Timothy L. Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proc. ACM Symp. on Operating Systems Principles (SOSP)*, 2003.

[6] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, Jan/Feb 2003.

[7] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL Server. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1997.

[8] Benoît Dageville and Mohamed Zaït. SQL memory management in Oracle9i. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2002.

[9] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. Symp. on Operating System Design and Implementation (OSDI)*, 2004.

[10] Karl Dias, Mark Ramacher, Uri Shaft, Venkateshwaran Venkataramani, and Graham Wood. Automatic performance diagnosis and tuning in Oracle. In *Proc. Conf. on Innovative Data Systems Research (CIDR)*, 2005.

[11] Said Elnaffar, Patrick Martin, and Randy Horman. Automatically classifying database workloads. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, 2002.

[12] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet Wiener, Armando Fox, Michael Jordan, and David Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2009.

[13] GoGrid. http://www.gogrid.com/.

[14] Hadoop. http://hadoop.apache.org/.

[15] James R. Hamilton. Cost of power in large-scale data centers, Nov 2008. http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx.

[16] Randy H. Katz. Tech titans building boom. *IEEE Spectrum*, Feb 2009.

[17] Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive control of virtualized resources in utility computing environments. In *Proc. European Conf. on Computer Systems (EuroSys)*, 2007.

[18] Piyush Shivam, Varun Marupadi, Jeffrey S. Chase, Thileepan Subramaniam, and Shivnath Babu. Cutting corners: Workbench automation for server benchmarking. In *Proc. USENIX Annual Technical Conference*, 2008.

[19] Ahmed A. Soror, Ashraf Aboulnaga, and Kenneth Salem. Database virtualization: A new frontier for database tuning and physical design. In *Proc. Workshop on Self-Managing Database Systems (SMDB)*, 2007.

[20] Ahmed A. Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, and Sunil Kamath. Automatic virtual machine configuration for database workloads. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2008.

[21] Adam J. Storm, Christian Garcia-Arellano, Sam Lightstone, Yixin Diao, and Maheswaran Surendra. Adaptive self-tuning memory in DB2. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2006.

[22] Virtual Computing Lab. http://vcl.ncsu.edu/.

[23] VMware. http://www.vmware.com/.

Privacy-preserving Digital Identity Management for Cloud Computing

Elisa Bertino
CS Department
Purdue University
West Lafayette, Indiana
bertino@cs.purdue.edu

Federica Paci
CS Department
Purdue University
West Lafayette, Indiana
paci@cs.purdue.edu

Rodolfo Ferrini
CS Department
Purdue University
West Lafayette, Indiana
rferrini@purdue.edu

Ning Shang
CS Department
Purdue University
West Lafayette, Indiana
nshang@cs.purdue.edu

Abstract

Digital identity management services are crucial in cloud computing infrastructures to authenticate users and to support flexible access control to services, based on user identity properties (also called attributes) and past interaction histories. Such services should preserve the privacy of users, while at the same time enhancing interoperability across multiple domains and simplifying management of identity verification. In this paper we propose an approach addressing such requirements, based on the use of high-level identity verification policies expressed in terms of identity attributes, zero-knolwedge proof protocols, and semantic matching techniques. The paper describes the basic techniques we adopt and the architeture of a system developed based on these techniques, and reports performance experimental results.

1 Introduction

Internet is not any longer only a communication medium but, because of the reliable, afforbable, and ubiquitous broadband access, is becoming a powerful computing platform. Rather than running software and managing data on a desktop computer or server, users are able to execute applications and access data on demand from the "cloud" (the Internet) anywhere in the world. This new computing paradigm is referred to as *cloud computing*. Examples of cloud computing applications are Amazon's Simple Storage Service (S3), Elastic Computing Cloud (EC2) for storing photos on Smugmug an on line photo service, and Google Apps for word-processing.

Cloud services make easier for users to access their personal information from databases and make it available to services distributed across Internet. The availability of such information in the cloud is crucial to provide

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

better services to users and to authenticate users in case of services sensitive with respect to privacy and security. Users have typically to establish their identity each time they use a new cloud service, usually by filling out an online form and providing sensitive personal information (e.g., name, home address, credit card number, phone number, etc.). This leaves a trail of personal information that, if not properly protected, may be misused. Therefore, the development of digital identity management (IdM for short) systems suitable for cloud computing is crucial. An important requirement is that users of cloud services must have control on which personal information is disclosed and how this information is used in order to minimize the risk of identity theft and fraud.

Another major issue concerning IdM in cloud platforms is interoperability. Interoperability issues range from the use of different identity tokens, such those encoded in X.509 certificates and SAML assertions, and different identity negotiation protocols, such as the client-centric protocols and the identity-provider centric protocols, to the use of different names for identity attributes. An *identity attribute* encodes a specific identity information about an individual, such as the social-security-number; it consists of an attribute name, also called identity tag, and a value. The use of different names for identity attributes, that we refer to as *naming heterogeneity*, typically occurs whenever users and cloud service providers use different vocabularies for identity attribute names. In this case, whenever a cloud service provider requests from a user a set of identity attributes to verify the user identity, the user may not understand which identity attributes he/she has to provide.

To address the problem of privacy-preserving management of digital identity attributes in domains with heterogeneous name spaces, we propose a privacy-preserving multi-factor identity attribute verification protocol supporting a matching technique based on look-up tables, dictionaries, and ontology mapping techniques to match cloud service providers and clients vocabularies. The protocol uses an aggregate zero knowledge proofs of knowledge (AgZKPK) cryptographic protocol to allow clients to prove with a single interactive proof the knowledge of multiple identity attributes without the need to provide them in clear.

The rest of the paper is organized as follows. Section 2 introduces the notions on which our multi-factor identity attribute verification protocol is based. Section 3 presents the multi-factor identity attribute verification protocol. Section 4 presents the system architecture and discuss implementation while Section 5 reports experimental results. Finally, Section 6 concludes the paper and outlines some future work.

2 Preliminary concepts

Our approach, as many other approaches, assumes an IdM system that include several entities: Identity Providers (IdPs), Cloud Service Providers (CSPs), Registars, and users. CSPs provide access to data and software that reside on the Internet. IdPs issue certified identity attributes to users and control the sharing of such information. Registrars are additional components that store and manage information related to identity attributes used in our multi-factor identity attribute verification approach. Note that, unlike the IdPs, the information stored at the Registrars does not include the values of the identity attributes in clear. Instead, such information only contains the cryptographic semantically secure commitments [1] of the identity attributes which are then used by the clients to construct zero knowledge proofs of knowledge (ZKPK) [2] of those attributes. The Registrar stores for each user an Identity Record (IdR) containing an identity tuple for each user's identity attribute m. Each identity tuple consists of a tag, that is, an attribute name, the Pedersen commitment [5] of m, denoted by M_i, the signature of the registrar on M, denoted by σ_i, two types of assurance, namely *validity assurance* and *ownership assurance*, and a set of nyms (also called weak identifiers) $\{W_{ij}\}$[3]. M_i is computed as $g^m h^r$, where m is the value of the

[1] A commitment scheme or a bit commitment scheme is a method that allows a user to commit to a value while keeping it hidden and preserving the user's ability to reveal the committed value later.

[2] A zero-knowledge proof or zero-knowledge protocol is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement.

[3] Nyms are used to link together different identity tuples of the same individual for multi-factor authentication. Nyms do not need to be protected.

identity attribute, r is a random number in $\mathcal{Z}_p$ and only known to the client, and g and h are generators of a group G of prime order p. G, g, h and p are public parameters of the Registrar. Validity assurance corresponds to the confidence about the validity of the identity attribute based on the verification performed at the identity attribute original issuer. Ownership assurance corresponds to the confidence about the claim that the principal presenting an identity attribute is its true owner. The identity tuples of each registered client can be retrieved from the Registrar by CSPs (*online* mode) or the Registrar can release to the client a certificate containing its identity record (*offline* mode).

3 Interoperable Multi-Factor Authentication

Our multi-factor authentication protocol takes place between a client and a CSP and consists of two phases. In the first phase, the CSP matches the identity attributes in the clients vocabulary with its own attributes to help the client understand its identity verification policy. An *identity verification policy* consists of the set of identity attributes that the user must prove to know; if the values of these identity attributes are only needed for verification purposes but not for the execution of the service required by the client, the CSP has no reason to have to see these values in clear. In the second phase, the client executes the AgZKPK protocol to prove the CSP the knowledge of the matched identity attributes. The use of this protocol allows the client to convince the CSP that the client knows the values of the identity attributes without having to reveal to the CSP the values in clear.

3.1 The protocol for identity attribute matching

Our attribute name matching technique uses a combination of look-up tables, dictionaries, and ontology mapping in order to address the different variations in identity attribute names. *Syntactic variations* refer to the use of different character combinations to denote the same term. An example is the use of “CreditCard” and “Credit_Card”. *Terminological variations* refer to the use of different terms to denote the same concept. An example of terminological variation is the use of the synonyms “Credit Card” and “Charge Card”. *Semantic variations* are related to the use of two different concepts in different knowledge domains to denote the same term. Syntactic variations can be identified by using look up tables. A look up table enumerates the possible ways in which the same term can be written by using different character combinations. Terminological variations can be determined by the use of dictionaries or thesaurus such as WordNet [6]. Finally, semantic variations can be solved by ontology matching techniques. An ontology is a formal representation of a domain in terms of concepts and properties relating those concepts. Ontologies can be used to specify a domain of interest and reason about its concepts and properties. Ontology mapping is the process whereby the concepts of an ontology - the source ontology - are mapped onto the concepts of another ontology - the target ontology - according to those semantic relations [4].

An important issue related to the identity matching protocol is which party has to execute the matching. In our approach the matching is performed by the CSP, in that performing the matching at the client has the obvious drawback that the client may lie and asserts that an identity attribute referred to in the CSP policy matches one of its attribute, whereas this is not the case. The use of ZKPK protocols (see next section) preserves the privacy of the user identity attributes by assuring that the CSP do not learn the values of these attributes; thus the CSP has no incentive to lie about the mapping. A second issue is how to take advantage of previous interactions that the client has performed with other CSPs. Addressing such issue is crucial in order to make the interactions between clients and CSPs fast and convenient for the users. To address such issue, the matching protocol relies on the use of *proof-of-identity certificates*; these certificates encode the mapping between (some of) the user identity attributes and the identity attributes referred in the policies of CSPs with which the user has successfully carried out past interactions.

Let *AttrProof* be the set of identity attributes that a CSP asks to a client in order to verify the identity of the user on behalf of which the client is running. Suppose that some attributes in *AttrProof* do not match any of the attributes in $AttrSet$, the set of clients' identity attributes. We refer to the set of component service's identity attributes that do not match a client attribute name to as *NoMatchingAttr*. The matching process consists of two main phases. The first phase matches the identity attributes that have syntactical and terminological variations. During this phase, the CSP sends to the client, for each identity attribute a_i in the *NoMatchingAttr* set, the set *Synset*$_i$ containing a set of alternative character combinations and a set of synonyms. The client verifies that for each identity attribute a_i, there is an intersection between *Synset*$_i$ and *AttrSet*. If this is the case attribute a_i is removed from *NoMatchingAttr*. Otherwise, if *NoMatchingAttr* is not empty, the second phase is performed. During the second phase the client sends *CertSet*, the set of its proof-of-identity certificates to the CSP. Thus, in the second phase of the matching process the CSP tries to match the concepts corresponding to the identity attributes the client is not able to provide with concepts from the ontologies of the CSPs which have issued the proof-of-identity certificates to the client. Only matches that have a confidence score s greater than a predefined threshold, set by the CSP, are selected. The greater the threshold, the greater is the similarity between the two concepts and thus higher is the probability that the match is correct. If the CSP is able to find mappings for its concepts, it then verifies by using the information in the proof-of-identity certificates that each matching concept matches a client's attribute *Attr*. If this check fails, the CSP terminates the interaction with the client.

3.2 Multi-factor authentication

Once the client receives *Match*, the set of matched identity attributes from the CSP, it retrieves from the Registrar or from its *RegCert*, that is a certificate repository local to the client, the commitments M_i satisfying the matches and the corresponding signatures σ_i. The client aggregates the commitments by computing $M = \prod_{i=1}^{n} M_i = g^{m_1+m_2+\ldots+m_i} h^{r_1+r_2+\ldots+r_i}$ and the signatures into $\sigma = \prod_{i=1}^{n} \sigma_i$, where σ_i is the Registrar 's signature on the committed value $M_i = g^{m_i} h^{r_i}$. According to the ZPK protocol, the client randomly picks y, s in $[1, ..q]$, computes $d = g^y h^s \pmod{p}$, and sends d, σ, M, M_i , $1 \leq i \leq t$, to the CSP. The CSP sends back a random challenge $e \in [1, .., q]$ to the client. Then the client computes $u = y + em \pmod{q}$ and $v = s + er \pmod{q}$ where $m = m_1 + \ \ldots \ m_t$ and $r = r_1 + \ \ldots \ r_t$ and sends u and v to the CSP. The CSP accepts the aggregated zero knowledge proof if $g^u h^v = dc^e$. If this is the case, the CSP then checks that $\sigma = \prod_{i=1}^{n} \sigma_i$. If also the aggregate signature verification succeeds, the CSP releases a proof-of-identity certificate to the client. The certificate states that client's identity attributes in the *Match* set are mapped onto concepts of the CSP ontology and that the client has successfully proved the knowledge of those attributes. The CSP sends the proof-of-identity certificate to the client and stores a copy of the certificate in its local repository *CertRep*. The proof-of-identity certificate can be can be provided to another CSP to allow the client to prove the knowledge of an attribute without performing the aggregate ZKP protocol. The CSP that receives the certificate has just to verify the validity of the certificate.

Example 1: Assume that a user Alice submits a request to her *Hospital Web portal* to access her test results. The *Hospital Web portal* retrieves the test results through the *Laboratory service*. The *Laboratory service* has to verify the identity of Alice in order to provide her test results. The identity verification policy of the *Laboratory service* requires Alice to provide *Medical Assurance*, *Social Security Number* and *Patient ID* identity attributes. Alice provides the aggregated proof of the required identity attributes to the *Hospital Web portal* which forwards them to the *Laboratory service*. The *Laboratory service* then verifies by carrying out an aggregate ZPK protocol with Alice that she owns the required attributes and releases a proof-of-identity certificate. Such certificate asserts Alice is the owner of the *Medical Assurance*, *Social Security Number* and *PatientID* identity attributes she has presented. The next time Alice would like to access her test results through *Hospital Web portal* portal she will present the proof-of-identity certificate to the *Hospital Web portal* which will forward the certificate to the *Laboratory service*. The *Laboratory service* will verify the validity of Alice's certificate and return the test results to the *Hospital Web portal* which will display the results to Alice.

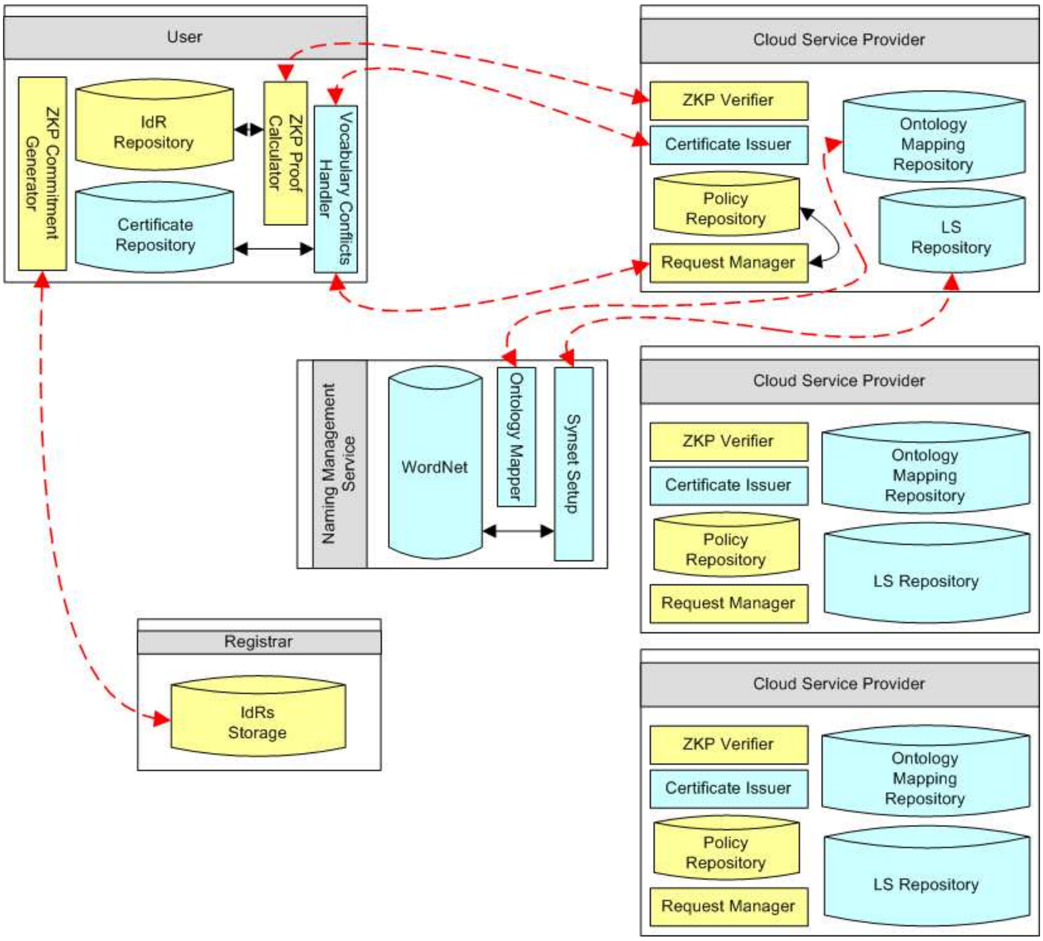


Figure 1: System architecture

4 System architecture and Implementation

In this section we discuss the system architecture that supports our multi-factor identity attributes authentication for cloud services. The architecture consists of four main components: the Registrar, the Service Provider, the User, and the Heterogeneity Management Service. The Registrar component manages the client's identity records and provide functions for retrieving the public parameters required by the AgZKPK protocol. The User component consists of three main modules: the ZKP Commitment Generator, the ZKP Proof Calculator, and the Vocabulary Conflicts Handler. The ZKP Commitment Generator provides the functions for computing the Pedersen commitments of identity attributes; the ZKP Proof Calculator generates the AgZKPK to be provided to CSPs, the Vocabulary Conflicts Handler module checks if there are client identity attributes names that matches the Synsets sent by the Service Provider component and manages the proof-of-identity certificates stored in a local repository. The Service Provider is composed of four modules the Request Manager, the Mapping Path Manager, the Certificate Issuer and the ZKP Verifier, and three repositories, one to store the mappings with other service provider ontologies, one to store the sets of synonymns Synsets, and one to store identity verification policies. The Request Manager component handles clients's requests and asks clients the identity attributes necessary for identity verification. The ZKP Verifier performs the AgZKPK verification. The Heterogeneity Management Services provides several functions shared by all CSPs. It consists of two modules: Synset SetUp and Ontology Manager. Synset SetUp returns the set of synonyms of a given term by querying a local thesaurus

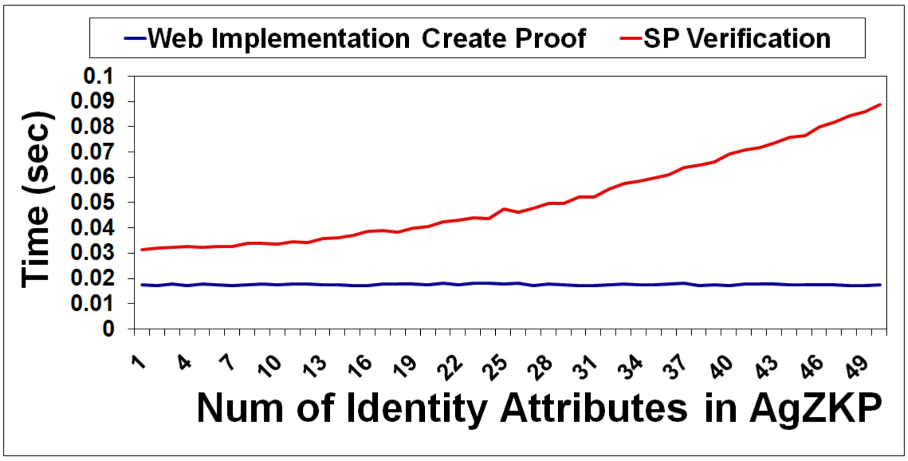


Figure 2: AgZKPK Verification versus Creation

while Ontology Manager provides the functionalities to perform the mapping between two ontologies.

The Service Provider application has been developed in JAVA. It implements the identity attribute name matching protocol using the Falcon-AO v0.7 [2, 3] ontology mapping API and WordNet 2.1 English Lexical database [6]. The User application has been implemented in JSP while the Registrar has been implemented as a JAVA servlet. Finally, we have used Oracle 10g DBMS to store clients' identity records, ontology mappings, set of synonyms, session data, and mapping certificates.

5 Experimental Evaluation

We have performed several experiments to evaluate the AgZKPK protocol that characterizes our approach to multi-factor identity verification and the identity attribute names matching process. We have carried out the following experimental evaluations:

- we have measured the time taken by the Client to generate the aggregate ZKP by varying the number of identity attributes being aggregated from 1 to 50;
- we have measured the time taken by the cloud service for aggregate ZKP verification execution time varying the number of identity attributes being aggregated from 1 to 50 (see Figure 2).

The execution time has been measured in CPU time (milliseconds). Moreover, for each test case we have executed twenty trials, and the average over all the trial execution times has been computed. Figure 2 reports the times to create an AgZKP and to verify it for varying values in the number of identity attributes being aggregated. The execution time to generate the AgZKP (represented by the blue line in the graph) is almost constant for increasing values in the number of identity attributes. The reason is that the creation of AgZKP only requires a constant number of exponentiations. By contrast, the time that the component Web service takes to perform identity attributes verification linearly increases with the number of identity attributes to be verified. The reason is that during the verification the component Web service is required to multiply all the commitments to verify the resulting aggregate signature.

6 Concluding Remarks

In this paper we have proposed an approach to the verification of digital identity for cloud platforms. Our approach uses efficient cryptographic protocols and matching techniques to address heterogeneous naming. We plan to extend this work in several directions. The first direction is to investigate the delegation of identity attributes from clients to CSPs. Delegation would allow a CSP, called the source CSP, to invoke the services of another CSP, called the receiving CSP, by passing to it the identity attributes of the client. However the receiving CSP must be able to verify such identity attributes in case it does not trust the source CSP. One possibility would be to allow the receiving CSP to directly interact with the client; however the source CSP may not be willing to allow the client to know the CSPs it uses for offering its services. Therefore protocols are needed able to address three requirements: confidentiality of business relations among the various CSPs, user privacy, and strenght of identity verification. The second direction is the investigation of unlinkability techniques. Our approach does not require that the values of the identity attributes only used for identity verification be disclosed to the CSPs; also our approach allows the user to use pseudonyms when interacting with the CSPs, if the CSP policies allow the use of pseudonyms and the user is interested in preserving his/her anonymity. However, if multiple transactions are carried out by the same user with the same CSP, this CSP can determine that they are from the same user, even if the CSP does not know who this user is nor the identity attributes of the user. Different CSPs may also collude and determine a profile of the transactions carried out by the same user. Such information when combined with other available information about the user may lead to disclosing the actual user identity or the values of some of his/her identity attributes, thus leading to privacy breaches. We plan to address this problem by investigating techniques that maintain unlinkability among multiple transactions carried out by the same user with the same or different CSPs.

7 Acknowledgments

This material is based in part upon work supported by the National Science Foundation under the ITR Grant No. 0428554 “The Design and Use of Digital Identities”, upon work supported by AFOSR grant A9550-08-1-0260, and upon work supported by the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001, under the auspices of the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security, the I3P, or Dartmouth College.

References

[1] Bhargav-Spantzel, A., Squicciarini, A.C, Bertino, E.: Establishing and Protecting Digital Identity in Federation Systems. Journal of Computer Security, IOSPress, 14(3), pp. 269–300, (2006)

[2] Choi, N., Song, I. Y., Han, H.: A survey on ontology mapping. SIGMOD Record 35, (3), pp. 34–41.

[3] Falcon, http://iws.seu.edu.cn/projects/matching/

[4] Y. Kalfoglou, and M. Schorlemmer. ”Ontology mapping: the state of the art.” The Knowledge Engineering Review, 18(1), pp. 1–31, (2003).

[5] Pedersen, T.P.: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. Advances in Cryptology, Proc. CRYPTO ’91, pp. 129–140, (1991).

[6] WordNet, http://wordnet.princeton.edu/

Towards a Scalable Enterprise Content Analytics Platform

Kevin Beyer   Vuk Ercegovac   Rajasekar Krishnamurthy   Sriram Raghavan   Jun Rao   Frederick Reiss
Eugene J. Shekita   David Simmen
Sandeep Tata   Shivakumar Vaithyanathan   Huaiyu Zhu

{*kbeyer, vercego, rajase, rsriram, junrao, frreiss, shekita, simmen, stata, vaithyan, huaiyu*}@*us.ibm.com*

IBM Almaden Research Center
650 Harry Road, San Jose
California, 95120, USA

Abstract

With the tremendous growth in the volume of semi-structured and unstructured content within enterprises (e.g., email archives, customer support databases, etc.), there is increasing interest in harnessing this content to power search and business intelligence applications. Traditional enterprise infrastruture or analytics is geared towards analytics on structured data (in support of OLAP-driven reporting and analysis) and is not designed to meet the demands of large-scale compute-intensive analytics over semi-structured content. At the IBM Almaden Research Center, we are developing an "enterprise content analytics platform" that leverages the Hadoop map-reduce framework to support this emerging class of analytic workloads. Two core components of this platform are SystemT, a high-performance rule-based information extraction engine, and Jaql, a declarative language for expressing transformations over semi-structured data. In this paper, we present our overall vision of the platform, describe how SystemT and Jaql fit into this vision, and briefly describe some of the other components that are under active development.

1 Introduction

As the volume of semi-structured and unstructured content within enterprises continues to grow, there is increasing interest and commercial value in harnessing this content to power the next generation of search and business intelligence applications. Some examples of enterprise repositories with valuable unstructured content include email archives, call center transcripts, customer feedback databases, enterprise intranets, and collaboration and document-management systems.

The use of content from such repositories for enterprise applications is predicated on the ability to perform analytics. For example, transcripts of customer calls can yield valuable business insights in areas such as product perception, customer sentiment, and customer support effectiveness. However, analytics is essential to extract the appropriate information from the raw text of the transcripts and transform this information into a form

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

that can be consumed by BI analysis and reporting applications. As another example, increased legislation around corporate data governance is requiring enterprises to invest in applications for regulatory compliance and legal discovery [5]. Such applications require advanced analytics (such as automatic recognition of persons, organizations, addresses, etc.) to support search and retrieval over enormous email archives. Finally, as we describe in Section 3, sophisticated analytics on intranet Web pages is critical to effective search over complex enterprise intranets.

Notice that in each of these applications, the common underlying theme is the need to perform analytics on large amounts of unstructured content. For instance, email archives can range in size from a few tens to several hundreds of terabytes, depending on the size of the company and the applicable regulations. Similarly, we know of close to 100M distinct URLs within the IBM intranet, which translates to approx. 3TB of plain HTML content. Factoring in non-HTML content as well as the data stored in numerous enterprise content management systems will result in a couple of orders of magnitude increase in size.

Traditional enterprise infrastructure for analytics (ETL engines, warehouses and marts, data cubes, etc.) is geared towards OLAP-driven reporting over structured data and is not designed to meet the demands of large-scale analytics over unstructured content. To this end, at the IBM Almaden Research Center, we are developing an *enterprise content analytics platform* to support muti-stage analytic workflows over large volumes of unstructured and semi-structured content. In building this platform, we are leveraging the tremendous innovation in the industry around scale-out based data processing paradigms – in particular, the open source Hadoop implementation of the Map/Reduce framework pioneered by Google.

The design goals for our platform are motivated by the following two observations:

1. *Compute intensive information extraction:* A common aspect of analytics over unstructured content is the need for *information extraction* – to identify and extract structured data ("annotations") from text. For example, in the ES2 intranet search application described in Section 3, several hundred patterns involving regular expressions and dictionary matches are applied on the titles, URLs, and other features of intranet Web pages to extract high quality annotations for a search index. In general, information extraction is a compute intensive process involving several complex character-level operations such as tokenization and pattern matching. The ability to support scalable information extraction over large content repositories is a key design goal for our platform.

2. *Dynamic and evolving workflows:* Due to the inherent heterogeneous nature of text collections, analytic workflows will need to continuously evolve over time to adapt to changes in the incoming content. For instance, when documents in newer formats or languages are added to a content source, appropriate modifications to the analytic workflow will be needed to incorporate new parsers, introduce language-specific tokenizers, and appropriately modify information extraction rule patterns. As a sample data point, the analytic workflow that powers the ES2 intranet search application (cf. Section 3) has gone through hundreds of changes to its processing workflow over the past year, in response to changes in the type and nature of content being published to the IBM intranet.

In this paper, we present a high-level architecture of our platform and describe some of the components that are currently under active development.

2 Overview of the Platform

The enterprise content analytics platform is designed to facilitate the specification and evaluation of complex analytic data flows over massive volumes (terabytes to petabytes) of content from enterprise repositories. We are developing new data processing tools for this purpose: SystemT [9] to extract information from content and Jaql [8] to flexibly specify analysis workflows. Both tools are used declaratively to insulate the user from optimization decisions needed for efficient and scalable processing.

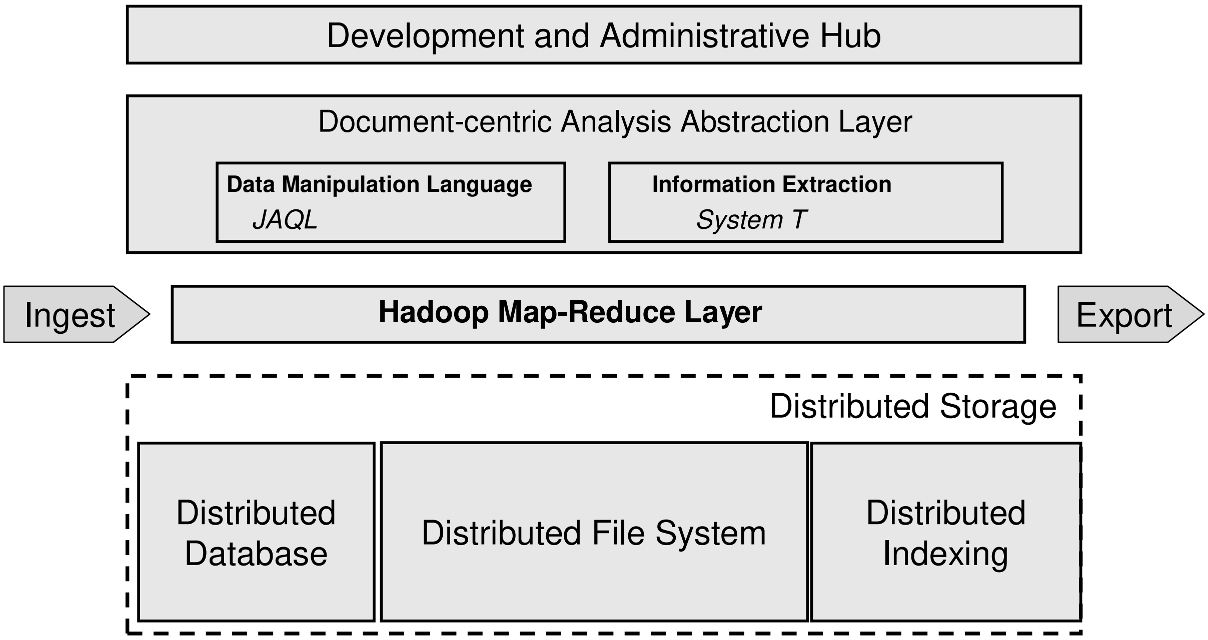


Figure 1: Architecture

In addition to SystemT and Jaql, the content analytics platform includes the additional tools shown in Figure 1. Prior to analysis, content is ingested and written to distributed storage services. For example, the semantic search application described in Section 3 uses Nutch [1], a distributed crawler based on Hadoop, to ingest data. In addition, we plan to ingest data in enterprise content management systems like FileNet, Documentum, and OpenText into the platform using special load APIs.

For scalable storage and processing, we have bootstrapped the content analytics platform using the Apache Hadoop [9] project's HDFS, a distributed file system, and map-reduce, a parallel programming framework popularized by Google [4]. Files provide the platform with scan-based access and map-reduce is used to implement parallel aggregations and joins by re-partitioning data across a cluster. In Section 4, we describe our active research and development efforts to extend the available storage services with a distributed database and index as well as extend map-reduce to incorporate query processing techniques from the database literature.

Following analysis, the results are transformed and exported according to application requirements. For example, a search application requires inverted indexes to be built. For those search applications that further expose structured information discovered during extraction, exporting to a relational database is appropriate. For such cases, we plan to support customizable export tools to accommodate varying application requirements.

While SystemT and Jaql are the core tools used for analysis tasks, we envision higher-level abstractions and tooling to assist users in developing analytic workflows. For content analytics, a *document* is the unit of analysis and therefore we are designing many of the operators (e.g., SystemT) to be document-centric. With regard to tooling, we plan to build a set of services and GUI to assist with analytics development, evaluation, and administration.

The current focus at the Almaden Research Center is on SystemT and Jaql. We now describe these in more detail.

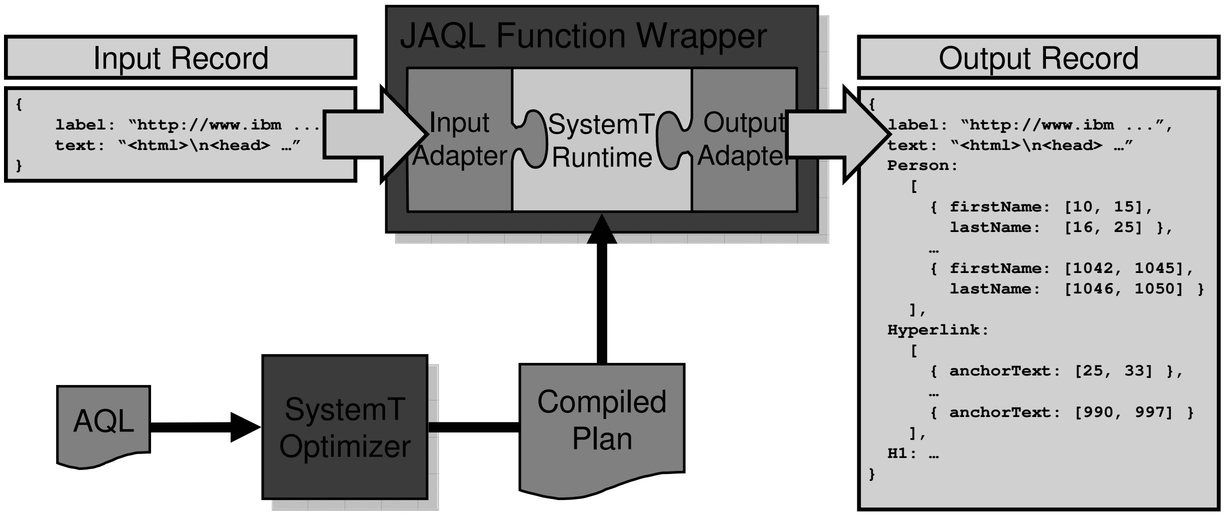


Figure 2: SystemT invoked from Jaql

2.1 Information Extraction using SystemT

SystemT is a system for rule-based information extraction that has been under development at IBM Almaden Research Center since 2006. SystemT is used to extract structured information from unstructured text, finding, for example, project home pages in corporate intranet web pages or person-phone number relationships in email messages. At the core of SystemT is a declarative rule language, AQL, for building extractors, and an optimizer that compiles these extractors for an algebra-based execution engine.

SystemT scales to massive document corpora by extracting information from multiple documents in parallel. The current version of the system supports two approaches to parallel scaleout: Direct embedding in a map-reduce job, and parallel execution as part of a Jaql query.

The direct embedding scaleout approach encapsulates the SystemT engine in a single "map" stage of a Hadoop map-reduce flow. The input to each mapper is a stream of documents, and the output of the mapper augments these documents with the structured information the system extracts from them. SystemT provides a layer of input/output adapters that support variety of input and output formats.

SystemT can also use Jaql's automatic parallelization to enable scalable information extraction (see Figure 2). SystemT's Jaql integration code encapsulates the information extraction runtime as a Jaql function. This function maps a record containing a document to an augmented record containing the document plus extracted structure. Jaql handles the mapping of the SystemT function call into a map-reduce framework, possibly executing a SystemT annotator and several other operations in a single map job.

2.2 Data Processing using Jaql

For flexibility during the ingestion, analysis, transformation, and exporting of data, we require a lightweight description language that supports semi-structured data and is easy to extend with new operators as well as sources and sinks of data. For this purpose, we are designing Jaql, a general purpose data-flow language that manipulates semi-structured information in the form of abstract JSON values.

JSON consists of atomic values like numbers and strings and two container types: arrays and records of name-value pairs (sometimes called maps or hashes). The values in a container are arbitrary JSON values; for example, arrays can have different types in each element and an element could itself be another array. The JSON

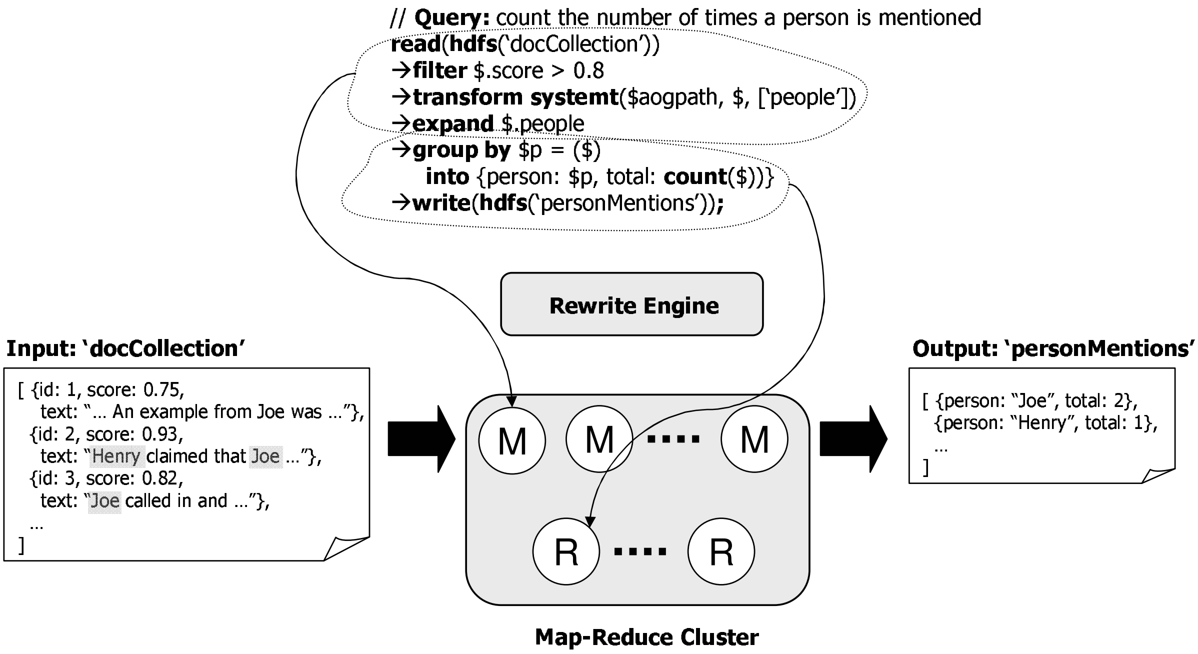


Figure 3: Jaql query compiled to map-reduce

model provides easy migration of data to and from most popular scripting languages like Javascript, Python, Perl, and Ruby because these languages all directly support dynamic arrays and records; other programming languages also have support for these constructs in their standard libraries. Therefore, Jaql is easily extended with operators written in most programming languages because JSON has a much lower impedance mismatch than say XML, yet much richer than relational tables.

Jaql provides a framework for reading and writing data in custom formats that is used while ingesting and exporting. We do not expect large volumes of JSON data to be stored on disk, but most data, like comma-separated files (CSVs), relational tables, or XML have a natural interpretation as JSON values. Unlike traditional database systems, Jaql processes data in its original format; the user is not required to load it into some internal form before manipulating it.

Jaql provides support for common input/output formats like CSVs, as well as many common operators including: filtering, transforming, sorting, grouping, aggregating, joining, merging, distincting, expanding nested data, and top-k. By composing these simple operators plus calls to custom operators into a rich data flow, the user expresses complex analyses and transforms the data to produce the exported data. Jaql compiles an entire flow of these operators into a graph of Map/Reduce jobs for Hadoop to process.

Figure 3 shows an example Jaql query which computes the number of times each person is mentioned in a given document collection. Using SystemT to detect mentions of a person's name, Jaql computes this count using the standard operator palette and produces the output in a JSON array.

3 Example Application: Semantic Search

One of the driving applications of the analytics platform at IBM is ES2, a semantic search engine for the enterprise. ES2 leverages the content analytics platform and uses sophisticated analytics along with an intelligent index-building strategy to provide high-precision results for navigational queries [10]. ES2 uses Nutch [1] as its ingest mechanism to crawl the pages on the IBM intranet. The pages crawled by Nutch are stored in HDFS and are available for processing using System T and Jaql. In this section, we briefly describe how System T and Jaql form critical building blocks for the analytics in a system like ES2.

Analytics The analysis in ES2 consists of two distinct types: local analysis which processes a single document at a time and global analysis which operates over the data extracted from the entire collection. The information obtained from these analyses is used as an input in the indexing phase. Together, local and global analyses allow us to reason about the document collection better and bring back significantly more precise results [10] than would be possible using traditional IR strategies such as *tf-idf* and *PageRank* [3].

In Local analysis each page is individually analyzed to extract clues that help decide whether that page is a candidate navigational page. In ES2, four different local analysis algorithms namely *TitleHomePage*, *PersonalHomePage*, *URLHomePage*, *AnchorHome* and *NavLink* are used to extract certain features from the pages. These algorithms use rules based on regular expression patterns, dictionaries, and information extraction tools [9] to identify candidate navigational pages. For instance, using a regular expression like “\ A\ W*(.+)\s<Home>” (Java regular expression syntax), the *PersonalHomePage* algorithm can detect that a page with a title “G. J. Chaitin’s Home” indicates that this is the home page of G. J. Chaitin. The algorithm outputs the name of a feature (“Personal Home Page”) and associates a value with this feature (“G. J. Chaitin”). Readers interested in more details about local analysis algorithms may refer to [10]. These analysis tasks are expressed using AQL and are optimized and executed using SystemT.

Note that multiple pages in the collection may produce the same extracted feature during local analysis. Consider the case where homepage authors use the same title for many of their webpages. Continuing the example from the previous paragraph, “GJ Chaitin home page” is the title for many of the pages on GJ Chaitin’s website. Local analysis for personal homepages considers all such pages to be candidates. ES2 uses global analysis to determine an appropriate subset of pages as navigational and indexes them appropriately. [10] describes two algorithms: *site root analysis* and *anchor text analysis*. In ES2, we express these algorithms using Jaql which automatically compiles into map-reduce jobs that execute over the entire cluster.

Smart Indexing ES2 employs a collection of algorithms to intelligently generate variants of terms extracted during local analysis and global analysis before inserting the document in the index. Consider an example where ES2 identifies a page as the home page of a person named “G J Chaitin” using local and global analysis. During variant generation, a special set of rules is applied to such person names to enumerate other syntactic variants (e.g., skipping middle initials, merging multiple initials, only listing the last name, etc.). By inserting such variants into the index, ES2 can match the search term “GJ Chaitin” with the given page, despite the lack of space between “G” and “J”. Note that generating variants by skipping white space, when applied to arbitrary pieces of text, is likely to yield noisy search results. We only apply this approach to pages produced through a specific analysis workflow – in this case one that uses results from *PersonalHomePage* local analysis and *site root* global analysis. Indexing is done in a distributed fashion by leveraging map-reduce. Jaql is used to transform the outputs of different analytic tasks and produce appropriate search indexes over the extracted values. The output from this workflow is typically a set of navigational indexes that are then copied over to a separate set of machines to serve queries.

4 Future Challenges

While SystemT and Jaql constitute a majority of our current effort, we are also investigating other components of the platform including the distributed runtime, distributed storage layer, and the user-interaction hub. We broadly outline the challenges in these areas.

4.1 Enhancing Map-reduce Runtime

The map-reduce paradigm was popularized by the distributed system community. Compared with a database system, map-reduce based data processing platforms have better support for fault-tolerance, elasticity, and load balancing. However, many computations in the two systems are quite similar. We are investigating techniques that bridge the gap between the two systems, by applying what the database community has learned over the last three decades to map-reduce. One of our areas of focus is to improve join processing in map-reduce.

Although map-reduce was originally designed to process a single collection of data, many analytic applications require joining multiple data sources together. Here is a straightforward way of mapping a join operation into map-reduce: the map function iterates over the records from both input sources. For each record, it extracts the join key as the output key, tags each record with the originating source, and saves it as the output value. Records from both sources for a given join key are eventually merged together and fed to a reduce task. The reduce function first separates the input records into two sets according to the tag, and then performs a cross-product between records in those sets. This implementation is similar to a repartitioned sort-merge join in a database system. This algorithm incurs a significant overhead since all input records have to be sorted and most of them have to be sent across the network.

For a long time, the database community has been exploiting hash-based joins to avoid sorting and broadcasting joins to avoid the communication overhead. Leveraging those ideas, an alternative approach is to do the join in a map-only job. At the beginning of each map task, we load the smaller input source into a hash table. The map function then iterates through records from the larger input source, and for each record, probes the hash table to do the join. This approach avoids sorting and moving data from the larger input source. Our experimental results show that it can reduce the time taken by the straightforward approach by up to 70%. As the smaller input source gets larger, the map-only job becomes less efficient since more data has to be broadcasted to every node. For certain applications, we find that semi-join techniques can be used to improve the performance further. We are preparing a research paper to summarize those results in detail. We are also investigating techniques to extend Jaql compiler to automatically select the best join strategy among the many available.

4.2 Distributed Content Database

Different phases of an analytics workflow tend to have different database requirements. For example, during ingestion, new documents are incrementally inserted into the database. Then during analysis, documents are often processed in batch mode as part of one or more map-reduce jobs. Finally, in applications like search, users often want to interactively look at documents returned by search queries. A key question is whether one content-oriented database is sufficient to satisfy all these requirements. With relational databases, it is common to maintain at least two databases, one for warehouse applications and another for interactive applications. We suspect that much the same will happen here. This is because content analytics tend to be very resource heavy, making it difficult to support interactive applications on the same database without running into performance problems.

Assuming separate content databases are maintained for analytics and interactive applications, then we think it will be interesting to explore different database architectures. One architecture would be tuned for batch analytics, while the other would be tuned for interactive applications. Generally speaking, interactive applications present a bigger design challenge in a distributed environment, especially if support for transactions are included. This is because of the well known tension between availability and consistency [2]. In contrast, analytics applications tend to work on a stable collection of documents, where consistency and availability tradeoffs are not an issue. Content-oriented databases without support for high-performance transactions like HBase [7] leverage the underlying distributed filesystem.

4.3 Development and Administrative Hub

We plan to build a development and administrative hub that facilitates the formation of user communities and management of resources on the platform. The hub would provide services for managing users (user services); for launching, monitoring, and diagnosing content analytics jobs (job services); for uploading and cataloging assets such as content analytic flows, data sources, data sinks, sandboxes, annotators, and user functions (directory services); and for performing miscellaneous tasks such as sandbox creation, load/unload of data to/from the cluster, and collection of data distribution statistics used to optimize analytic flows (utility services).

Providing a design interface that allows workflows to be created iteratively, and interactively, in the context of a ”sandbox” is one of the challenges being tackled. A sandbox includes representative cluster configuration information, as well as representative samples from relevant data sources. We are also examining tools that would assist in the collaborative development of analytic components.

5 Summary

With increasing interest from enterprises to harness the value in their structured and semi-structured content, we believe that there is a rapidly emerging need for an enterprise content analytics platform. We identify two key features that are needed from such a platform: 1) the ability to perform compute intensive information extraction, and 2) build and maintain evolving workflows that process large amounts of data. We describe the efforts underway at IBM’s Almaden Research Center to address these requirements by way of SystemT and Jaql. In addition, we also laid out the broad challenges that lay ahead in building a dynamic, scalable, high-performance, and usable content analytics platform for enterprises.

References

[1] Apache Foundation. Nutch. http://lucene.apache.org/nutch/.

[2] Eric Brewer. Keynote speech: Towards robust distributed systems. In *PODC*, 2000.

[3] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107 – 117, 1998. Proceedings of the Seventh International World Wide Web Conference.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[5] Electronic Discovery. Electronic discovery. http://www.discoveryresources.org/.

[6] Apache Foundation. Hadoop. http://hadoop.apache.org/core/.

[7] Apache Foundation. Hbase. hadoop.apache.org/hbase.

[8] JAQL. Jaql. http://code.google.com/p/jaql/.

[9] Frederick Reiss, Sriram Raghavan, Rajasekar Krishnamurthy, Huaiyu Zhu, and Shivakumar Vaithyanathan. An algebraic approach to rule-based information extraction. In *ICDE*, pages 933–942, 2008.

[10] Huaiyu Zhu, Sriram Raghavan, Shivakumar Vaithyanathan, and Alexander Löser. Navigating the intranet with high precision. In *WWW*, pages 491–500, 2007.

Building a Cloud for Yahoo!

Brian F. Cooper, Eric Baldeschwieler, Rodrigo Fonseca, James J. Kistler, P.P.S. Narayan,
Chuck Neerdaels, Toby Negrin, Raghu Ramakrishnan, Adam Silberstein,
Utkarsh Srivastava, and Raymie Stata

Yahoo! Inc.

Abstract

Yahoo! is building a set of scalable, highly-available data storage and processing services, and deploying them in a cloud model to make application development and ongoing maintenance significantly easier. In this paper we discuss the vision and requirements, as well as the components that will go into the cloud. We highlight the challenges and research questions that arise from trying to build a comprehensive web-scale cloud infrastructure, emphasizing data storage and processing capabilities. (The Yahoo! cloud infrastructure also includes components for provisioning, virtualization, and edge content delivery, but these aspects are only briefly touched on.)

1 Introduction

Every month, over half a billion different people check their email, post photos, chat with their friends, and do a myriad other things on Yahoo! sites. We are constantly innovating by evolving these sites and building new web sites, and even sites that start small may quickly become very popular. In addition to the websites themselves, Yahoo! has built services (such as platforms for social networking) that cut across applications. Sites have typically solved problems such as scaling, data partitioning and replication, data consistency, and hardware provisioning individually.

In the cloud services model, all Yahoo! offerings should be built on top of cloud services, and only those who build and run cloud services deal directly with machines. In moving to a cloud services model, we are optimizing for human productivity (across development, quality assurance, and operations): it should take but a few people to build and rapidly evolve a Web-scale application on top of the suite of horizontal cloud services. In the end-state, the bulk of our effort should be on rapidly developing application logic; the heavy-lifting of scaling and high-availability should be done in the cloud services layer, rather than at the application layer, as is done today. Observe that while there are some parallels with the gains to be had by building and re-using common software platforms, the cloud services approach goes an important step further: developers are insulated from the details of provisioning servers, replicating data, recovering from failure, adding servers to support more load, securing data, and all the other details of making a neat new application into a web-scale service that millions of people can rely on.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In this paper, we describe the requirements, how the pieces of the cloud fit together, and the research challenges, especially in the areas of data storage and processing. We note that while Yahoo!'s cloud can be used to support externally-facing cloud services, our first goal is to provide a common, managed, powerful infrastructure for Yahoo! sites and services, i.e., to support internal developers. It is also our goal to open source as many components of the cloud as possible. Some components (such as Hadoop) are already in open source. This would allow others outside of Yahoo! to build their own cloud services, while contributing fixes and enhancements that make the cloud more useful to Yahoo!

2 Requirements

Yahoo! has been providing several centrally managed data management services for years, and while these services are not properly "cloud services" they have many of the characteristics. For example, our database of user profiles is run as a central service. Accessing the user database requires only the proper permissions and a client library, avoiding the need to set up and manage a separate user information repository for every application. Experience with these "proto-cloud" services have helped inform the set of requirements we laid out for our cloud:

Multitenancy We must be able to support many applications (tenants) on the same hardware and software infrastructure. These tenants must be able to share information but have their performance isolated from one another, so that a big day for Yahoo! Mail does not result in a spike in response time for Yahoo! Messenger users. Moreover, adding a new tenant should require little or no effort beyond ensuring that enough system capacity has been provisioned for the new load.

Elasticity The cloud infrastructure is sized based on the estimates of tenant requirements, but these requirements are likely to change frequently. We must be able to quickly and gracefully respond to requests from tenants for additional capacity, e.g., a growing site asks for additional storage and throughput.

Scalability We must be able to support very large databases, with very high request rates, at very low latency. The system should be able to scale to take on new tenants or handle growing tenants without much effort beyond adding more hardware. In particular, the system must be able to automatically redistribute data to take advantage of the new hardware.

Load and Tenant Balancing We must be able to move load between servers so that hardware resources do not become overloaded. In particular, in a multi-tenant environment, we must be able to allocate one application's unused or underused resources to another to provide even faster absorption of load spikes. For example, if a major event is doubling or quadrupling the load on one of our systems (as the 2008 Olympics did for Yahoo! Sports and News), we must be able to quickly utilize spare capacity to support that extra load.

Availability The cloud must always be on. If a major component of the cloud experiences an outage, it will not just be a single application that suffers but likely all of them. Although there may be server or network failures, and even a whole datacenter may go offline, the cloud services must continue to be available. In particular, the cloud will be built out of commodity hardware, and we must be able to tolerate high failure rates.

Security A security breach of the cloud will impact all of the applications running on it; security is therefore critical.

Operability The systems in the cloud must be easy to operate, so that a central team can manage them at scale. Moreover, the interconnections between cloud systems must also be easy to operate.

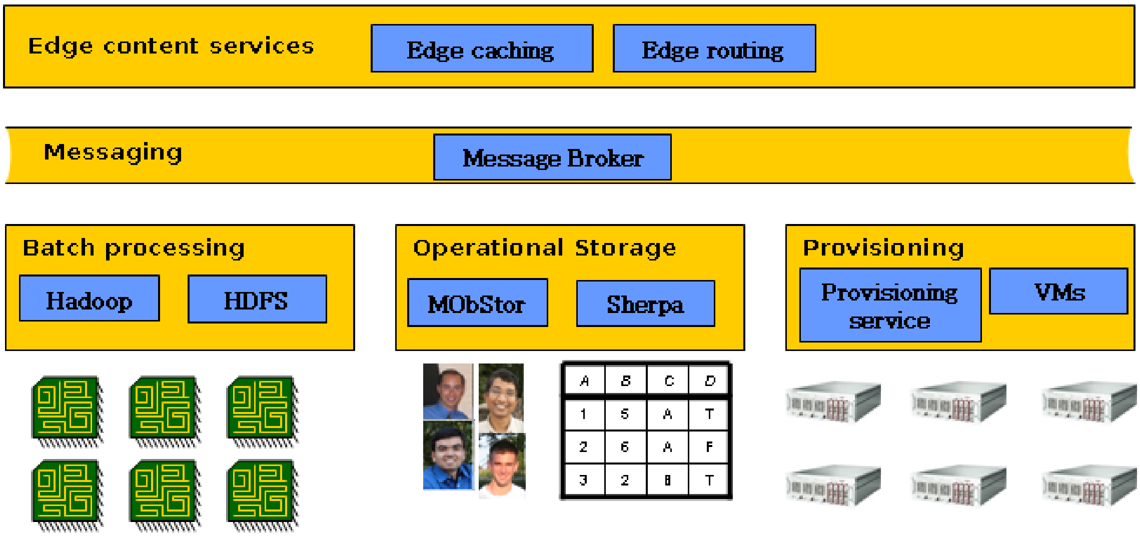


Figure 1: Components of the Yahoo! data and processing cloud.

Metering We must be able to monitor the cloud usage of individual applications. This information is important to make provisioning decisions. Moreover, the cloud will be paid for by those applications that use it, so usage data is required to properly apportion cost.

Global Yahoo! has users all over the world, and providing a good user experience means locating services in datacenters near our users. This means that cloud services must span continents, and deal with network delays, partitions and bottlenecks as they replicate data and services to far flung users.

Simple APIs We must expose simple interfaces to ease the development cost of using the cloud, and avoid exposing too many parameters that must be tuned in order for tenant applications to get good performance.

3 Overall architecture

Yahoo!'s cloud focuses on "horizontal services," which are common platforms shared across a variety of applications. Those applications may themselves be "vertical services," which are task-specific applications shared by a variety of end users. For example, we view Yahoo! Mail as a vertical service, while a blob store (such as our MObStor system) is a horizontal service that can store attachments from Mail, photos from Flickr, movie trailers from Yahoo! Movies, and so on.

Figure 1 shows a block diagram of the main services in our cloud. As the figure shows, there are three tiers of services: core services; messaging; and edge services. While the bottom tier provides the heavy lifting for server-side data management, the edge services help reduce latency and improve delivery to end users. These edge services include edge caching of content as well as edge-aware routing of requests to the nearest server and around failures. The messaging tier helps tie disparate services together. For example, updates to an operational store may result in a cache invalidation, and the messaging tier carries the invalidation message to the cache.

The bottom tier of core services in Figure 1 is further subdivided into three groups of systems. Batch processing systems manage CPU cycles on behalf of large parallel jobs. Specifically, we have deployed Hadoop, an open source version of MapReduce [3], and its HDFS filesystem. Operational storage systems manage the storage and querying of data on behalf of applications. Applications typically have two kinds of operational data: structured records and unstructured blobs. In our infrastructure, structured data is managed by Sherpa (also known as PNUTS [2]), while blobs are stored in MObStor. Provisioning systems manage the allocation of servers for all of the other service components. One way to provision servers is to deploy them as virtual machines, and our provisioning framework includes the ability to deploy either to a VM or to a "bare" machine.

The horizontal services in our cloud provide platforms to store, process and effectively deliver data to users. A typical vertical application will likely combine multiple horizontal services to satisfy all of its data needs. For example, Flickr might store photos in MObStor and photo tags in Sherpa, and use Hadoop to do offline analysis to rank photos in order of popularity or "interestingness." The computed ranks may then be stored back in Sherpa to be used when responding to user requests. A key architecture question as we move forward deploying the cloud is how much of this "glue" logic combining different cloud services should be a part of the cloud as well.

In the rest of this article, we focus on the operational storage and batch computation components, and examine these components in more detail.

4 Pieces of the cloud

4.1 Hadoop

Hadoop [1] is an open source implementation of the MapReduce parallel processing framework [3]. Hadoop hides the details of parallel processing, including distributing data to processing nodes, restarting subtasks after a failure, and collecting the results of the computation. This framework allows developers to write relatively simple programs that focus on their computation problem, rather than on the nuts and bolts of parallelization. Hadoop data is stored in the Hadoop File System (HDFS), an open source implementation of the Google File System (GFS) [4].

In Hadoop, developers write their MapReduce program in Java, and divide the logic between two computation phases. In the Map phase, an input file is fed to a task, which produces a set of key-value pairs. For example, we might want to count the frequency of words in a web crawl; the map phase will parse the HTML documents and output a record `(term,1)` for each occurrence of a term. In the Reduce phase, all records with the same key are collected and fed to the same reduce process, which produces a final set of data values. In the term frequency example, all of the occurrences of a given term (say, "cloud") will be fed to the same reduce task, which can count them as they arrive to produce the final count.

The Hadoop framework is optimized to run on lots of commodity servers. Both the MapReduce task processes and the HDFS servers are horizontally scalable: adding more servers adds more compute and storage capacity. Any of these servers may fail at any time. If a Map or Reduce task fails, it can be restarted on another live server. If an HDFS server fails, data is recovered from replicas on other HDFS servers. Because of the high volume of inter-server data transfer necessary for MapReduce jobs, basic commodity networking is insufficient, and extra switching resources must be provisioned to get high performance.

Although the programming paradigm of Hadoop is simple, it enables many complex programs to be written. Hadoop jobs are used for data analysis (such as analyzing logs to find system problems), data transformation (such as augmenting shopping listings with geographical information), detecting malicious activity (such as detecting click fraud in streams of ad clicks) and a wide variety of other activities.

In fact, for many applications, the data transformation task is sufficiently complicated that the simple framework of MapReduce can become a limitation. For these applications, the Pig language [5] can be a better framework. Pig provides relational-style operators for processing data. Pig programs are compiled down to Hadoop MapReduce jobs, and thus can take advantage of the scalability and fault tolerance of the Hadoop framework.

4.1.1 Hadoop in the cloud

Hadoop runs on a large cluster of centrally managed servers in the Yahoo! cloud. Although users can download and run their own Hadoop instance (and often do for development purposes) it is significantly easier to run Hadoop jobs on the centrally managed processing cluster. In fact, the convenience of storing and processing data in the cloud means that much of the data in our cluster is Hadoop from birth to death: the data is stored in HDFS at collection time, processed using MapReduce, and delivered to consumers without being stored in another filesystem or database. Other applications find it more effective to transfer their data between Hadoop and another cloud service. For example, a shopping application might receive a feed of items for sale and store them in Sherpa. Then, the application can transfer large chunks of listings to Hadoop for processing (such as geocoding or categorization), before being stored in Sherpa again to be served for web pages.

Hadoop is being used across Yahoo by multiple groups for projects such as response prediction for advertising, machine learned relevance for search, content optimization, spam reduction and others. The Yahoo! Search Webmap is a Hadoop application that runs on a more than 10,000 core Linux cluster and produces data that is now used in every Yahoo! Web search query. This is the largest Hadoop application in production, processing over a trillion page links, with over 300 TB of compressed data. The results obtained were 33 percent faster than the pre-Hadoop process on a similar cluster. This and a number of other production system deployments in Yahoo! and other organizations demonstrate how Hadoop can handle truly Internet scale applications in a cost-effective manner[1].

4.2 MObStor

Almost every Yahoo! application uses mass storage to store large, unstructured data files. Examples include Mail attachments, Flickr photos, restaurant reviews for Yahoo! Local, clips in Yahoo! Video, and so on. The sheer number of files that must be stored means they are too cumbersome to store and organize on existing storage systems; for example, while a SAN can provide enough storage, the simple filesystem interface layered on top of a SAN is not expressive enough to manage so many files. Moreover, to provide a good user experience, files should be stored near the users that will access them.

The goal of MObStor is to provide a scalable mass storage solution. The system is designed to be scalable both in terms of total data stored, as well as the number of requests per second for that data. At its core, MObStor is a middleware layer which virtualizes mass storage, allowing the underlying physical storage to be SAN, NAS, shared nothing cluster filesystems, or some combination of these. MObStore also manages the replication of data between storage clusters in geographically distributed datacenters. The application can specify fine-grained replication policies, and the MObStor layer will replicate data according to the policies.

Applications create collections of files, and each file is identified with a URL. This URL can be embedded directly in a web page, enabling the user's browser to retrieve files from the MObStore system directly, even if the web page itself is generated by a separate HTTP or application server. URLs are also virtualized, so that moving or recovering data on the back end filesystem does not break the URL. MObStor also provides services for managing files, such as expiring old data or changing the permissions on a file.

4.2.1 MObStor in the cloud

As with the other cloud systems, MObStor is a centrally managed service. Storage capacity is pre-provisioned, and new applications can quickly create new collections and begin storing and serving data. Mobstor uses a flat domain based access model. Applications are given a unique domain and can organize their data in any format they choose. A separate metadata store provides filesystem-like semantics: users can create, list and delete files through the REST interface.

[1]Thanks to Ajay Anand from the Hadoop team for these statistics.

Mobstor is optimized for serving data to internet users at Yahoo! scale. A key component of the architecture is a caching layer that also supports streaming. This enables the system to offload hot objects to the caching infrastructure, allowing the I/O subsystem to scale. Like other cloud services, Mobstor strives to locate data close to users to reduce latency. However due to the cost of the underlying storage and the fact that users are less sensitive to latency with large files, Mobstor does not have to support the same level of replication as the other cloud services.

There are many Yahoo! applications currently using MObStor. Examples include:

- Display ads for the APT platform
- Tile images for Yahoo! Maps
- Files shared between Yahoo! Mail users
- Configuration files for some parts of the Yahoo! homepage
- Social network invites for the Yahoo! Open platform

Each of these use cases benefits from the ability to scalably store and replicate a large number of unstructured objects and to serve them with low latency and high throughput.

4.3 Sherpa

The Sherpa system, also called PNUTS in previous publications [2, 6], presents a simplified relational data model to the user. Data is organized into tables of records with attributes. In addition to typical data types, "blob" is a valid data type, allowing arbitrary structures inside a record, but not necessarily large binary objects like images or audio; MObStor is a more appropriate store for such data. We observe that blob fields, which are manipulated entirely in application logic, are used extensively in practice. Schemas are flexible: new attributes can be added at any time without halting query or update activity, and records are not required to have values for all attributes. Sherpa allows applications to declare tables to be hashed or ordered, supporting both workloads efficently.

The query language of Sherpa supports selection and projection from a single table. We designed our query model to avoid operations (such as joins) which are simply too expensive in a massive scale system. While restrictive compared to relational systems, our queries in fact provide very flexible access that covers most of the web workloads we encounter. The system is designed primarily for online serving workloads that consist mostly of queries that read and write single records or small groups of records. Thus, we expect most scans to be of just a few tens or hundreds of records, and optimize accordingly. Scans can specify predicates which are evaluated at the server. Similarly, we provide a "multiget" operation which supports retrieving multiple records (from one or more tables) in parallel by specifying a set of primary keys and an optional predicate, but again expect that the number of records retrieved will be a few thousand at most.

While selections can be by primary key or specify a range, updates and deletes must specify the primary key. Consider a social networking application: A user may update her own record, resulting in access by primary key. Another user may scan a set of friends in order by name, resulting in range access.

Data in Sherpa is replicated to globally distributed datacenters. This replication is done asynchronously: updates are allowed to a given replica, and success is returned to the user before the update is propagated to other replicas. To ensure the update is not lost, it is written to multiple disks on separate servers in the local datacenter.

Asynchronously replicated data adds complexity for the developer. Sherpa provides a consistency model to simplify the details of reading and writing possibly stale data, and to hide the details of which replica is being accessed from the applciation.

4.3.1 Sherpa in the cloud

Sherpa is a hosted service, and the software and servers are managed by a central group. Applications that wish to use Sherpa can develop against a single-server standalone instance. However, all production data is served from cloud servers. This allows application developers to focus on their application logic, and leave the details of designing, deploying and managing a data architecture to a specialized group. In order to support this hosted model, the Sherpa operations group must provision enough capacity to support all the applications that will use it. Currently, we work with customers to estimate their capacity needs and then pre-provision servers for their use. We are moving to a model with extra servers in a "free pool," and if an application's load on Sherpa begins to increase, we can automatically move servers from the free pool into active use for that application.

Sherpa is designed to work well with the other cloud services. For example, Hadoop can use Sherpa as a data store instead of the native HDFS, allowing us to run MapReduce jobs over Sherpa data. We also have implemented a bulk loader for Sherpa which runs in Hadoop, allowing us to transfer data from HDFS into a Sherpa table. Similarly, Sherpa can be used as a record store for other cloud services. As an example, MObStor is investigating using Sherpa to store metadata about files.

5 Open questions

Many research questions have arisen as we build the cloud, both at the level of individual components and across components. In this section, we discuss some key questions that span cloud components. Although many of our cloud components are in production or nearing completion, these questions will have to be resolved in order for the cloud to reach its full potential.

Interacting with the cloud How do users interact with the cloud? One possibility is that each application chooses individual cloud systems, and manages their interactions at the application level. For example, suppose a user has a record-oriented data set and an OLTP workload. He therefore loads it into a Sherpa database. Periodically, he does extensive OLAP work. At these times, he loads the data set from Sherpa to HDFS, and runs Hadoop jobs.

However, one of the advantages of using the cloud is that it can provide seamless integration between multiple services. The job of the developer is easier if he does not have to explicitly manage data storage. For applications that will use multiple services, a nicer abstraction may be that data is placed "in the cloud" and is accessible to any service the application needs. In the above example, the data may be stored in Sherpa, but when an OLAP job is submitted, the cloud software decides whether to move the data to HDFS or to run a MapReduce job directly over Sherpa. This approach makes the cloud more complex, as it is an optimization problem which must take into account the query workload as well as information about the current capabilities and load on each of the services; the profile of future queries from the same application; the load from other services; and so on.

Another way we can make it easier for developers to use cloud services is to provide a common API for the various services. For example, we may develop a query language which spans multiple services (e.g., some combination of Pig, SQL, and the simple Sherpa and MObStor access languages) which then compiles to operations on individual services as well as actions to move data between them. If we hide many of the data placement and service selection decisions behind a declarative query language, we may not always make the best decisions without at least some input from the developer. We will likely need a mechanism for profiling the performance of the system, so that developers can readily identify which components are becoming a bottleneck. Such a mechanism will need to monitor an entire application as it interacts across multiple cloude services. In addition, a hint mechanism will be needed which allows the application to guide data placement and allocation decisions based on observations of the bottlenecks.

Quality of service A key question for any shared infrastructure is how we can isolate the performance of different applications. Applications will place variable load on the cloud, and a spike in one application's workload will affect other applications sharing the same hardware. One approach to this problem is to place quotas on applications' resource usage. This approach is too inflexible, since spare resources cannot be used to absorb load spike beyond an application's quota. We could also use some version of weighted fair sharing (like that used in networking systems), which allows spare resources to be allocated to needy applications. However, the infrastructure needed to monitor and dynamically allocate resources is complex. A third approach is to have a small number (e.g. two) of application classes. "Gold" applications can use as many resources as they like, while "bronze" applications are served by the remaining resources in a best effort manner. This moves resource allocation decisions into the hands of the business people, who must carefully choose just a few gold applications. Whatever approach we use will have to effectively enforce QoS for an application, even as it crosses over between Sherpa, Hadoop and MObStor, and other components of the cloud.

Other open issues There are several other issues which we are investigating:

- *Automating operations* - Our central operations group will be managing many servers, many different services, and many applications. Tools and processes which automate things like failover and resource allocation will make their jobs significantly easier.
- *Growth* - Applications which start small can grow to become quite large. Although our cloud services are designed to scale elastically, we must investigate how well they tolerate growth of one, two or more orders of magnitude.
- *Privacy* - Each system in the cloud has its own data model and thus its own model of enforcing privacy for that data. When data starts moving between systems, we must ensure that the change to a different data model does not cause a privacy breach. Moreover, we must ensure that multi-tenant applications can only see each other's data if doing so does not violate the user's privacy.
- *Capacity management* - It is difficult to know how much hardware to dedicate to the cloud to meet the anticipated load. Even if one resource (e.g. CPU) is plentiful, another resource (e.g. in-memory cache space) may be scarce. Similarly, a shift in the load, such as a move from sequential to random record access, can create a new bottleneck in a previously plentiful resource. We need to develop effective and comprehensive models for planning the capacity needs of the cloud.

6 Conclusion

We believe that the Yahoo! cloud will be a key to both lower costs and increased innovation. The Cloud Computing and Data Infrastructure division has the charter to develop cloud services such as Hadoop, MObStor and Sherpa, make them elastic, robust and reliable, and integrate them into a comprehensive cloud infrastructure. Many cloud components are already adopted widely, and we are seeing further rapid growth on the horizon.

References

[1] Hadoop. hadoop.apache.org.

[2] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!s hosted data serving platform. In *Proc. VLDB*, 2008.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proc. SOSP*, 2003.

[5] C. Olston et al. Pig Latin: A not-so-foreign language for data processing. In *Proc. SIGMOD*, 2008.

[6] A. Silberstein et al. Efficient bulk insertion into a distributed ordered table. In *Proc. SIGMOD*, 2008.

On the Varieties of Clouds for Data Intensive Computing

Robert L. Grossman
University of Illinois at Chicago
and Open Data Group

Yunhong Gu
University of Illinois at Chicago

Abstract

By a cloud *we mean an infrastructure that provides resources or services over a network, often the Internet, usually at the scale and with the reliability of a data center. We distinguish between clouds that provide on-demand computing instances (such as Amazon's EC2 service) and clouds that provide on-demand computing capacity (such as provided by Hadoop). We give a quick overview of clouds and then describe some open source clouds that provide on-demand computing capacity. We conclude with some research questions.*

1 Introduction

1.1 Types of Clouds

There is not yet a standard definition for cloud computing, but a good working definition is to say that *clouds* provide on demand resources or services over a network, often the Internet, usually at the scale and with the reliability of a data center.

There are quite a few different types of clouds and again there is no standard way of characterizing the different types of clouds. One way to distinguish different types of clouds is to categorize the architecture model, computing model, management model and payment model. We discuss each of these below. See Table 1.

1.2 Architectural Model

We begin with the architecture model. There are at least two different, but related, architectures for clouds: the first architecture is designed to provide computing *instances* on demand, while the second architecture is designed to provide computing *capacity* on demand.

Amazon's EC2 services [1] provides computing instances on demand and is an example of the first architectural model. A small EC2 computing instance costs $0.10 per hour and provides the approximate computing power of 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor, with 1.7 GB memory, 160 GB of available disk space and moderate I/O performance [1].

Google's MapReduce provides computing capacity on demand and is an example of the second architectural model for clouds. MapReduce was introduced by Google in the paper [8]. This paper describes a sorting application that was run on a cluster containing approximately 1800 machines. Each machine had two 2 GHz Intel Xeon processors, 4 GB memory, and two 160 GB IDE disks. The TeraSort benchmark [10] was coded

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Model	Variants
Architecture Model	clouds that provide on-demand computing instances; clouds that provide on-demand computing capacity
Programming Model	Using queues to pass message; MapReduce over storage clouds; UDFs over storage clouds; message passing
Management Model	private vs shared; internal vs hosted
Payment Model	pay as you go; subscribe for a specified period of time; buy

Table 1: Some different types of clouds.

using MapReduce, a parallel programming model which is described in more detail below. The goal of the TeraSort benchmark is to sort 10^{10} 100-byte records, which is about 1 TB of data. The application required about 891 seconds to complete [8] on this cluster.

The Eucalyptus system [19] is an open source cloud that provides on demand computing instances and shares the same APIs as Amazon's EC2 cloud. The Hadoop system is an open source cloud that implements a version of MapReduce [16].

Notice that both types of clouds consist of loosely coupled commodity computers, but that the first architecture is designed to scale out by providing additional computing instances, while the second architecture is designed to support data or compute intensive applications by scaling computing capacity. By scaling computing capacity, we mean the ability to aggregate a large number of loosely coupled computers so that the aggregate infrastructure can manage very large datasets, sustain very large aggregate input/output, and perform a very large aggregate number of computing tasks.

1.3 Programming Model

Clouds that provide on-demand computing instances can support any computing model compatible with loosely coupled clusters. For example, instances in an Amazon EC2 can communicate using web services [3], using queues [2], or using message passing. It is important to note though that the performance using message passing on loosely coupled systems is much slower than message passing or tightly coupled clusters.

Clouds that provide on-demand computing capacity can also support any computing model compatible with loosely coupled clusters. Programming using web services and message passing can be complicated though and beginning with [8], a programming model called MapReduce has become the dominant programming model used in clouds that provide on-demand computing capacity. MapReduce assume that many common programming applications can be coded as processes that manipulate large datasets consisting of $<$key, value$>$ pairs. Map is a process that maps each $<$key, value$>$ pair in the dataset into a new pair of $<$key$'$, value$'>$. Reduce is a process that merges values with the same key. Although this is a seemingly simple model, it has been used to support a large number of data intensive applications, especially applications that must manipulate web related data. MapReduce is described in more detail in Section 2.1 below.

Stream-based parallel programming models in which a User Defined Function (UDF) is applied to all the data managed by the cloud have also proved to be quite useful [14].

1.4 Payment Model

Amazon popularized a cloud that provides on-demand computing instances with a "pay as you go" economic model. By simply setting up a Amazon Web Services account that links to a credit card, one can set up a computing instance, with attached storage and network connectivity and pay about 10 cents an hour for just those hours that you actually use the resources.

Of course, you can also buy, set up, and run your own cloud. Alternately, you can make arrangements with a third party to pay for the exclusive use of cloud resources for a specified period of time.

1.5 Management Model

The hardware for clouds can be by provided internally by an organization (internal clouds) or externally by a third party (hosted clouds). A cloud may be restricted to a single organization or group (private clouds) or shared by multiple groups or organizations (shared clouds). All combinations of these management options arise.

1.6 What's New?

Local and remote loosely coupled clusters have been available for quite some time and there is a large amount of middleware available for such clusters. Because of this, it is important to ask what is new with clouds.

The first thing that is new is the scale. Google and Yahoo have reported computing on clouds that contain 1000, 2000 and up to 10,000 loosely coupled computers. With Hadoop, datasets that are tens to hundreds of terabytes can be managed easily, something that requires significant effort with a database.

The second thing that is new is the simplicity that clouds provide. For example, with just a credit card and a browser connected to the Internet, you can use Amazon's EC2, S3, and SQS to bring up 100 computing instances, perform a computation, and return the results without any capital investment, without hiring a system administrator, and without installing and mastering any complex middleware. Useful machine images containing precisely the pre-installed software required can be invoked by simply referencing an Amazon Machine Image identifier, such as ami-3c47a355.

As another example, with MapReduce, a new software engineer can be analyzing a 10 TB dataset of web data on 100 nodes with less than a day of instruction by using simple, small MapReduce programs.

It is interesting to note that this style of cloud computing came from industry's need for a simple to use, yet powerful platform for high performance computing, not from academic research in high performance computing.

2 Clouds That Provide On-Demand Computing Capacity

2.1 Google's Storage, Compute and Table Cloud Services

The basic architecture for clouds that provide on-demand computing capacity was articulated in a series of Google technical reports. See Figure 1. A cloud storage service called the Google File System (GFS) was described in [9]. GFS was designed to scale to clusters containing thousands of nodes and was optimized for appending and for reading data.

For computing with data managed by GFS, a parallel programming framework for loosely coupled systems called MapReduce was described in [8]. A good way to describe MapReduce is through an example: Assume that node i in a cloud stores web pages $p_{i,1}$, $p_{i,2}$, $p_{i,3}$, ..., $p_{i,n}$. Assume also that web page p_i contains words w_1, w_2, w_{i_j}, ... A basic structure important in information retrieval is an inverted index, which is a data structure consisting of a word followed by a list of web pages

$$(w_1; p_{1,1}, p_{2,1}, p_{3,2}, \ldots)$$

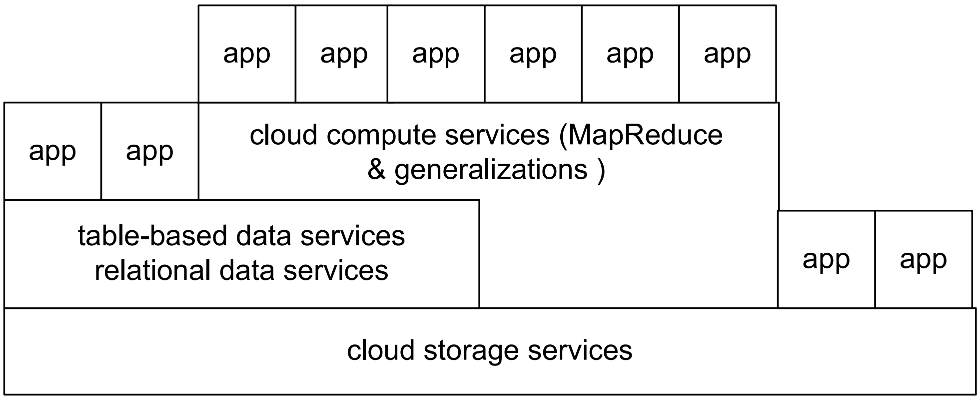


Figure 1: Clouds that provide on-demand computing capacity often layer services as shown in the diagram.

$$(w_2; p_{1,1}, p_{1,2}, p_{3,1}, \ldots)$$

$$(w_3; p_{1,3}, p_{2,2}, p_{3,3}, \ldots)$$

with the properties:

1. The inverted index is sorted by the word w_j;

2. If a word w_j occurs in a web page p_i, then the web page p_i is on the list associated with the word w_j.

A mapping function processes each web page independently, on its local storage node, providing data parallelism. The mapping function emits multiple <key, value> pairs (<keyword, page_id> in this example) as the outputs. This is called the Map Phase.

A partition function $m(w)$, which given a word w, assigns a machine labeled with $m(w)$, is then used to send the outputs to multiple common locations for further processing. This second step is usually called the Shuffle Phase.

In the third step, the processor $m(w_i)$ sorts all the <key, value> pairs according to the key. (Note that there may be multiple keys sent to the same node, i.e., $m(w_i) = m(w_j)$.) Pairs with same key (keyword in this example) are then merged together to generate a portion of the inverted index $<w_i$: $p_{x,y}, \ldots)>$. This is called the Reduce Phase.

To use MapReduce, a programmer simply defines the (input) Record Reader (for parsing), Map, Partition, Sort (or Comparison), and Reduce functions and the infrastructure takes care of the rest.

Since many applications need access to rows and columns of data (not just bytes of data provided by the GFS), a GFS-application called BigTable [5] that provides data services that scale to thousands of nodes was developed. BigTable is optimized for appending data and for reading data. Instead of the ACID requirements of traditional databases, BigTable choose an eventual consistency model.

2.2 Open Source Clouds That Provide On-Demand Computing Capacity

The Google's GFS, MapReduce and BigTable are proprietary and not generally available. Hadoop [16] is an Apache open source cloud that provides on-demand computing capacity and that generally follows the design described in the technical reports [9] and [8]. There is also an open source application called HBase that runs over Hadoop and generally follows the BigTable design described in [8].

Sector is another open source system that provides on-demand computing capacity [18]. Sector was not developed following the design described in the Google technical reports, but instead was designed to manage and distribute large scientific datasets, especially over wide area high performance networks. One of the first

Design Decision	Google's GFS, MapReduce, BigTable	Hadoop	Sector
data management	block-based file system	block-based file system	data partitioned into files; native file system used
communication	TCP	TCP	UDP-Based Data Transport (UDT) and SSL
programming model	MapReduce	MapReduce	User defined functions, MapReduce
replication strategy	at the time of writing	at the time of writing	periodically
security	not mentioned	yes	yes (HIPAA capable)
language	C++	Java	C++

Table 2: Some of the similarities and differences between Google's GFS and MapReduce, Hadoop and Sector.

Sector applications was the distribution of the 10+ TB Sloan Digital Sky Survey [15]. Sector is based upon a network protocol called UDT that is designed to be fair and friendly to other flows (including TCP flows), but to use all the otherwise available bandwidth in wide area high performance network [13].

The Hadoop Distributed File System (HDFS), like Google's GFS, implements a block-based distributed file system, which is a fairly complex undertaking. HDFS splits files to into large data blocks (usually 64MB each) and replicates each block on several nodes (the default is to use three replicas). In contrast, Sector assumes that the user has split a dataset into several files, with the size and number of files depending upon the number of nodes available, the size of the dataset, and the anticipated access patterns. Although this imposes a small burden on the user, the result is a much simpler design can be used for the underlying system.

On top of the Sector Distributed File System is a parallel programming framework that can invoke user defined functions (UDFs) over the data managed by Sector. Three specific, but very important UDFs, are the Map, Shuffle and Reduce UDFs described above, which are available in Sector.

Table 2 contains a summary of some of these similarities and differences.

2.3 Experimental Studies

In this section, we describe some experimental studies comparing the performance of Sector and Hadoop. The experiments were performed on the Open Cloud Testbed, a testbed managed by the Open Cloud Consortium [17]. The Open Cloud Testbed consists of four geographically distributed racks located in Chicago (two locations), San Diego and Baltimore and connected by 10 Gb/s networks. Each contains 30 Dell 1435 computers with 4GB memory, 1TB disk, 2.0GHz dual-core AMD Opteron 2212, with 1 Gb/s network interface cards. Since the tests were done, the current equipment in the Open Cloud Testbed has been upgraded and and additional sites have been added.

Table 3 contains some experimental studies comparing Sector and Hadoop using the Terasort benchmark [10]. The tests placed 10GB of data on each node. The tests were run on a single rack, two racks connected by a Metropolitan Area Network in Chicago, three racks connected by a Wide Area Network, and four racks connected by a Wide Area Network. In all cases, the networks provided 10 Gb/s of bandwidth. Notice that although there is a penality incurred for the computing across geographhically distributed racks, it is not prohibitive. It is about 20% when using Sector and about 64% when using Hadoop, when wide area high performance networks are available.

Table 4 contains some experimental studies that were done using CreditStone [4], which is a benchmark that can be used for testing clouds that provide on-demand computing capacity. CreditStone provides code that generates synthetic events that are roughly modeled on credit card transactions and flags some of the transactions.

	Number of nodes	Sector	Hadoop
WAN-2 (UIC, SL, UCSD, JHU)	118	3702 sec	1526 sec
WAN-1 (UIC, SL, UCSD)	88	3069 sec	1430 sec
MAN (UIC, SL)	58	2617 sec	1301 sec
LAN (UIC)	29	2252 sec	1265 sec

Table 3: The table shows the time required to complete the Terasort benchmark. The tests were run on the Open Cloud Testbed. The time required to generate the data is excluded. The test used 10 GB of data per node. The four racks on the testbed were connected by a 10 Gb/s network.

# Locations	Sector	Hadoop	# Events
1 location, 30 nodes, LAN	36 min	126 min	15 billion
4 locations, 117 nodes, WAN	71 min	189 min	58.5 billion

Table 4: Some experimental studies using the CreditStone benchmark comparing Hadoop and Sector run on the Open Cloud Testbed. Hadoop was configured to use one replica for these experiments.

The benchmark requires that certain ratios of unflagged to flagged transactions be computed, a computation that is quite straightforward to do using MapReduce, UDFs, or similar programming models.

3 Research Questions

In this section, we discuss several research questions.

1. In Section 2, we discussed two parallel programming models for clouds that provide on-demand computing capacity (MapReduce and invoking UDFs on dataset segments managed by a storage cloud), both of which are more limited than parallel programming using message passing but which most programmers find easier to use. A research question is to investigate other parallel programming models for these types of clouds that cover a different class of applications but are also quite easy to use.

2. Most clouds today are designed to do the computation within one data center. A interesting research question is to develop appropriate network protocols, architectures and middleware for wide area clouds that span multiple data centers.

3. Another research question is to investigate how different clouds can interoperate; that is, how two different clouds, perhaps managed by two different organizations, can share information.

4. A practical question is to develop standards and standards based architectures for cloud services for clouds that provide on-demand computing capacity so, for example, alternate storage, compute, or table services could be used in a cloud application.

References

[1] Amazon. Amazon Elastic Compute Cloud (amazon ec2). ams.amazon.com/ec2, 2008.

[2] Amazon. Amazon Seb Services Queue Service. Retrieved from http://aws.amazon.com/sqs, 2008.

[3] Amazon. Amazon Web Services Developer Connection. Retrieved from http://aws.amazon.com, 2008.

[4] Collin Bennett, Robert L Grossman, Jonathan Seidman, and Steve Vejcik. Creditstone: A benchmark for clouds that provide on-demand capacity. to appear, 2008.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *OSDI'06: Seventh Symposium on Operating System Design and Implementation*, 2006.

[6] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, volume 19, 2007.

[7] Data Mining Group. Predictive Model Markup Language (pmml), version 3.2. http://www.dmg.org, 2008.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, 2004.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM.

[10] Jim Gray. Sort benchmark home page. http://research.microsoft.com/barc/SortBenchmark/, 2008.

[11] Robert L Grossman and Yunhong Gu. Data mining using high performance clouds: Experimental studies using sector and sphere. In *Proceedings of The 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008)*. ACM, 2008.

[12] Robert L Grossman, Mark Hornick, and Gregor Mayer. Data mining standards initiatives. *Communications of the ACM*, 45(8):59–61, 2002.

[13] Yunhong Gu and Robert L Grossman. UDT: UDP-based data transfer for high-speed wide area networks. *Computer Networks*, 51(7):1777—1799, 2007.

[14] Yunhong Gu and Robert L Grossman. Sector and sphere: Towards simplified storage and processing of large scale distributed data. *Philosophical Transactions of the Royal Society A, also arxiv:0809.1181*, 2009.

[15] Yunhong Gu, Robert L Grossman, Alex Szalay, and Ani Thakar. Distributing the sloan digital sky survey using udt and sector. In *Proceedings of e-Science 2006*, 2006.

[16] Hadoop. Welcome to Hadoop! hadoop.apache.org/core/, 2008.

[17] Open Cloud Consortium. http://www.opencloudconsortium.org, 2009.

[18] Sector. http://sector.sourceforge.net, 2008.

[19] Rich Wolski, Chris Grzegorczyk, and Dan Nurmi et. al. Eucalyptus. retrieved from http://eucalyptus.cs.ucsb.edu/, 2008.

Optimizing Utility in Cloud Computing through Autonomic Workload Execution

Norman W. Paton, Marcelo A. T. de Aragão, Kevin Lee, Alvaro A. A. Fernandes, Rizos Sakellariou
School of Computer Science, University of Manchester, U.K.
(norm,maragao,klee,rizos,alvaro)@cs.man.ac.uk

Abstract

Cloud computing provides services to potentially numerous remote users with diverse requirements. Although predictable performance can be obtained through the provision of carefully delimited services, it is straightforward to identify applications in which a cloud might usefully host services that support the composition of more primitive analysis services or the evaluation of complex data analysis requests. In such settings, a service provider must manage complex and unpredictable workloads. This paper describes how utility functions can be used to make explicit the desirability of different workload evaluation strategies, and how optimization can be used to select between such alternatives. The approach is illustrated for workloads consisting of workflows or queries.

1 Introduction

Cloud computing essentially provides services; shared computational resources execute potentially diverse requests on behalf of users who may have widely differing expectations. In such a setting, someplace in the architecture, decisions have to be made as to which requests from which users are to be executed on which computational resources, and when. From the perspective of the service provider, such decision making may be eased through the provision of restrictive interfaces to cloud services, as discussed for cloud data services in the Claremont Report on Database Research [1]:

> Early cloud data services offer an API that is much more restricted than that of traditional database systems, with a minimalist query language and limited consistency guarantees. This pushes more programming burden on developers, but allows cloud providers to build more predictable services, and to offer service level agreements that would be hard to provide for a full-function SQL data service. More work and experience will be needed on several fronts to explore the continuum between today's early cloud data services and more full-functioned but probably less predictable alternatives.

This paper explores part of this space, by describing an approach to workload execution that is applicable to different types of workload and that takes account of: (i) the properties of the workload; (ii) the nature of the service level agreement associated with user tasks; and (iii) competition for the use of finite, shared resources.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

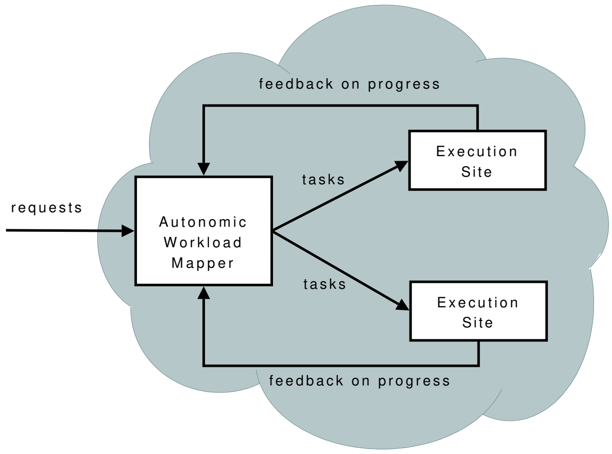


Figure 1: High level architecture.

In so doing, we explore functionalities that in future may be supported within a cloud, rather than by layering rich application functionality over lean cloud interfaces, as in Brantner *et al.* [4].

Wherever services are provided, service users have expectations; Service Level Agreements (SLAs) make explicit what expectations users can realistically place on a service provider [16], and may be associated with a charging model that determines the remuneration associated with certain Qualities of Service (QoS). Whether or not formal agreements are in place, decisions must nonetheless be made that influence the behaviors users experience, and service providers must put in place mechanisms that make such decisions.

In this paper, we assume the abstract architecture illustrated in Figure 1, where an *autonomic workload mapper* provides workload evaluation services implemented within a cloud. For the moment, we are non-specific about the nature of these services, but in due course details will be provided about support for workloads consisting of collections of queries or workflows. The *autonomic workload mapper* adaptively assigns tasks in the workload to execution sites. Given some objective, such as to minimize total execution times or, more generally, to optimize for some QoS target (whether these objectives are imposed by an SLA or not), the autonomic workload mapper must determine which tasks to assign to each of the available execution sites, revising the assignment during workload execution on the basis of feedback on the overall progress of the submitted requests.

In this paper, we investigate the use of *utility functions* [9] to make explicit the desirability of the state of a system at a point in time. In essence, a utility function maps each possible state of a system to a common scale; the scale may represent response times, numbers of QoS goals met, income based on some charging model for the requests, etc. In this setting, it is the goal of the *autonomic workload mapper* to explore the space of alternative mappings with a view to maximizing utility as measured by the utility function. We propose that utility functions, combined with optimization algorithms that seek to maximize utility for a workload given certain resources, may provide an effective paradigm for managing workload execution in cloud computing.

The remainder of this paper is structured as follows. Section 2 describes a methodology for developing utility based autonomic workload execution. Sections 3 and 4 describe the application of the methodology to workloads consisting of sets of workflows and queries, respectively. Section 5 presents some conclusions.

2 Utility Driven Workload Execution

When a utility-based approach is adopted, the following steps are followed by designers; instantiations of each of these steps are detailed for workloads consisting of workflows and queries in Sections 3 and 4, respectively.

Utility Property Selection: Identify the property that it would be desirable to maximize – useful utility measures may be cast in terms of response time, number of QoS targets met, profit, etc.

Utility Function Definition: Define a function $Utility(w, a)$ that computes the utility of an assignment a of tasks to execution sites for a workload w expressed in terms of the chosen property – for workload mapping, such a function can be expected to include expressions over variables v_e that describe the environment and the assignment a that characterizes the mapping for the components of w from abstract requests to tasks executing on specific execution sites.

Cost Model Development: Develop a cost model that predicts the performance of the workload given the information about the environment v_e and assignment a, taking into account the costs associated with adaptations.

Representation Design: Design a representation for the assignment a of workload components to computational resources, where adaptations to the assignment can be cast as modifications to this representation. For example, if a workload consists of a collection of tasks, then an assignment a of tasks to sites may be represented as a vector v where each element v_i represents task i, and each element value represents the execution site to which the task is assigned.

Optimization Algorithm Selection: Select an optimization algorithm that, given values for v_e, searches the space of possible assignments a with a view to maximizing the utility function; one benefit of the utility-based approach is that standard optimization algorithms can be used to explore the space of alternative mappings. Note that one benefit of the methodology is that it decouples the problem of meeting certain objectives under certain constraints into a modeling problem (i.e., to come up with a utility function) and an optimization problem (where standard mathematical techniques can be used).

Control Loop Implementation: Implement an autonomic controller [8] that: *monitors* the progress of the workload and/or properties of the environment of relevance to the utility function; *analyses* the monitored information to identify possible problems or opportunities for adaptation; *plans* an alternative workload execution strategy, by seeking to maximize $Utility(w, a)$ in the context of the monitored values for v_e; and *updates* the workload execution strategy where planning has identified an assignment that is predicted to increase utility.

Several researchers have reported the use of utility functions in autonomic computing, typically to support systems management tasks (e.g. [19, 3]); to the best of our understanding this is the first attempt to provide a methodology for the use of utility functions for adaptive workload execution.

3 Autonomic Workflow Execution

A cloud may host computational services in a specific domain; for example, the CARMEN e-Science cloud provides a collection of data and analysis services for neuroscience, and applications are constructed using workflow enactment engines hosted within the cloud [20]. In such a setting, autonomic workflow execution must determine how best to map workflows to the resources provided by the cloud.

3.1 Problem Statement

A workload w consists of a set of workflow instances i, each of which consists of a collection of tasks, $i.tasks$, and is evaluated through an allocation of tasks to a set of execution sites. The role of the autonomic workload mapper is to adaptively assign the tasks to specific sites.

3.2 Methodology Application

The methodology from Section 2 can be applied in this example as follows.

Utility Property Selection: Two utility properties are considered here, namely *response time* and *profit*. In practice, a single utility function is used by an autonomic workload mapper, but alternatives are shown here to illustrate how the approach can be applied to address different system goals.

Utility Function Definition: A utility function is defined for each of the properties under consideration. For *response time* we have:

$$Utility_w^{RT}(w, a) = 1/(\Sigma_{i \in w} PRT_w(i, a_i))$$

where, w is the set of workflows, a is a set of assignments for the workflows instances i in w, a_i is the assignment for workflow instance i, and PRT_w estimates the predicted response time of the workflow for the given assignment.

For *profit* we have:

$$Utility_w^{Profit}(w, a) = \Sigma_{i \in w}(Income(i, a_i) - EvaluationCost(i, a_i))$$

where $Income$ estimates the income that will be received as a result of evaluating i using allocation a_i, and $EvaluationCost(w, a)$ estimates the financial cost of the resources used to evaluate i. In this utility function, we assume that income is generated by evaluating workflows within a response time target, but that an $EvaluationCost$ is incurred for the use of the resources to evaluate the workflows. As the income depends on the number of QoS targets met, which in turn depends on reponse time, the definition of $Income$ is defined in terms of PRT_w. In cloud computing, the evaluation cost could reflect the fact that at times of low demand all requests can be evaluated using (inexpensive) resources within the cloud, but that at times of high demand it may be necessary to purchase (expensive) cycles from another cloud in order to meet QoS targets.

Cost Model Development: The cost model must implement $PRT_w(i, a_i)$; the predicted response time of a workflow depends on the predicted execution times of each of the tasks on their assigned execution site, the time taken to move data between execution sites, the other assignments of workflows in w, etc. The description of a complete cost model is beyond the scope of this paper, but cost models for workflows have been widely studied (e.g. [12, 18, 21]).

Representation Design: For each workflow instance $i \in w$, the assignment of the tasks $i.tasks$ can be represented by a vector v where each element v_i represents task i, and each element value represents the execution site to which the task is assigned.

Optimization Algorithm Selection: The optimization algorithm seeks to maximize $Utility(w, a)$ by exploring the space of alternative assignments a. As the assignments are represented as collections of categorical variables, each representing the assignment of a task to a specific execution site, an optimization algorithm must be chosen for searching such discrete spaces (e.g. [2]).

Control Loop Implementation: In autonomic workflow management [13], there is a requirement to halt an existing workflow, record information on the results produced to date, deploy the revised workflow in such a way that it can make use of results produced to date, and continue with the evaluation.

In practice, the utility functions described above prioritize different behaviors, and effective optimization can be expected to yield results that reflect those priorities. For example, $Utility_w^{RT}(w, a)$ will always seek the fastest available solution, even if this involves the use of costly computational resources. As a result, $Utility_w^{Profit}(w, a)$ will typically yield response times that are slower than those obtained by $Utility_w^{RT}(w, a)$, as it will only use expensive resources when these are predicted to give net benefits when considered together with the income they make possible. A detailed description of utility-based workflow execution in computational grids, including an experimental comparison of behaviors exhibited by different utility functions, is given by Lee *et al.* [12].

4 Autonomic Query Workload Execution

Early cloud data services are typically associated with fairly restrictive data access models with a view to enabling predictable behaviors, and do not provide full query evaluation [1]. However, more comprehensive data access services could provide access either to arbitrary query evaluation capabilities or to parameterized queries, thus giving rise to a requirement for query workload management, where collections of query evaluation requests can be managed by [11]: (i) an *admission controller*, which seeks to identify and disallow access to potentially problematic requests; (ii) a *query scheduler*, which determines when jobs are released from a queue for execution; and (iii) an *execution controller*, which determines the level of resource allocated to queries while they are executing. In this paper we discuss how utility functions can be used to direct the behavior of an *execution controller*. In comparison with recent work on workload management, a utility-driven approach can provide relatively fine-grained control over queries; for example, Krompass *et al.* [10] describe an execution controller in which the actions carried out at query runtime are job-level (i.e., reprioritize, kill and resubmit), whereas here the optimization makes global decisions (taking into account all the queries in the workload) that adaptively determine the resource allocations of individual queries on the basis of (fine-grained, collected per query) progress and load data.

4.1 Problem Statement

A workload w consists of a set of queries $q \in w$, each of which are evaluated on a collection of execution sites, potentially exploiting both partitioned and pipelined parallelism. Each query is associated with a distribution policy $dp(q)$, of the form $[v_1, v_2, \ldots, v_{|S|}]$, where $0 \leq v_i \leq 1$ and $(\Sigma_{i=1}^{|S|} v_i) \in \{0, 1\}$ where $|S|$ is the number of available execution sites. If the sum of v_i yields 1 then each v_i represents the fraction of the workload that is to be evaluated on the ith site using partitioned parallelism, and if the sum is 0 this represents the suspension of the plan. Where v_i is 0 for some i this represents the fact that execution site i is not being used for q. The role of the autonomic workload mapper in Figure 1 is to adaptively compute distribution policies for each of the queries in the workload.

4.2 Methodology Application

The methodology from Section 2 can be applied in this example as follows.

Utility Property Selection: Two utility properties are considered here, namely *response time* and *number of QoS targets met*. In the second case, we assume that each query is associated with a response time target.

Utility Function Definition: A utility function is defined for each of the properties under consideration. For *response time* we have:

$$Utility_q^{RT}(w, dp) = (1/\Sigma_{q \in w} PRT_q(q, dp(q)))$$

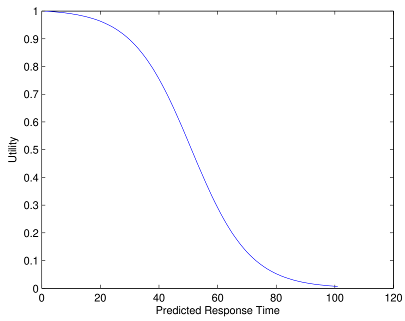


Figure 2: $QoSEstimate$ for a target response time of *50*.

where, w is the set of queries, dp is a distribution policy for the queries $q \in w$, and PRT_q estimates the predicted response time of the query for the given distribution policy.

For *quality of service* we have:

$$Utility_q^{QoS}(w, a) = \Sigma_{q \in w} QoSEstimate(q, dp(q))$$

where $QoSEstimate(q, dp(q))$ estimates the likelihood that the query will meet its QoS target using the given distribution policy from its predicted response time PRT_q. In practice, $QoSEstimate(q, dp(q))$ can be modeled using a curve such as that illustrated in Figure 2, which gives a score near to 1 for all queries estimated to be significantly within the target response time, and a score near to 0 for all queries estimated to take significantly longer than their target response time [3].

Cost Model Development: The cost model must implement $PRT_q(q, dp(q))$ for queries during their evaluation, and can build on results on query progress monitors (e.g. [5, 7]).

Representation Design: For each query $q \in w$, the distribution policy can be represented by a vector v where each element v_i represents the fraction of the work for q that is to be assigned to execution site i.

Optimization Algorithm Selection: The optimization algorithm seeks to maximize the utility function by exploring the space of distribution policies dp. As the assignments are represented as fractions, each representing the portion of the work to be assigned to a specific execution site, an optimization technique must be chosen for searching such spaces (e.g., sequential quadratic programming [6]).

Control Loop Implementation: The implementation of the control loop must be able to suspend an evaluating query, relocate operator state to reflect changes to the distribution policy, and continue evaluation using the updated plan. A full description of such a protocol is provided in the paper on Flux [17].

As an example of the behaviors exhibited by workflow execution management techniques, we have experimentally evaluated several such techniques using a simulator of a parallel query evaluation engine [15]. Figure 3 shows results for five different strategies: *No Adapt*, in which no runtime adaptation takes place; *Adapt 1* in which workloads are managed using action based control strategies (i.e. *if-then* rules based on Flux [17]) that seek to minimize response times by adapting whenever load imbalance is detected; *Adapt 2* in which utility functions are used to minimize response times, as in $Utility_q^{RT}$; *Adapt 3* in which *Adapt 2* is applied only when it is predicted that response time targets will be missed; and *Adapt 4* in which which utility functions are used to maximize the number of response time targets met, as in $Utility_q^{QoS}$. In this experiment, four queries each containing a single join are submitted at the same time to a cluster containing *12* execution sites, where one

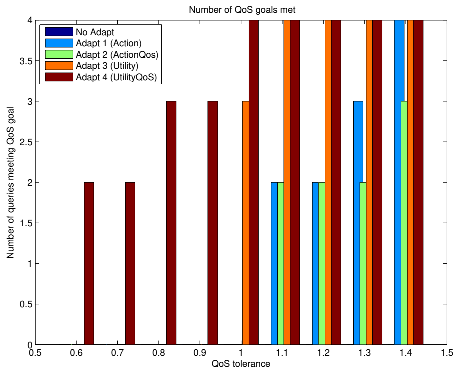


Figure 3: Numbers of Quality of Service Targets met by adaptive techniques [15].

of the sites is subject to periodic interference from other jobs broadly half of the time. In the experiment, the queries are associated with varying QoS targets (shown on the horizontal axis, with the more stringent targets to the left), and the number of queries meeting their reponse time targets is illustrated on the vertical axis.

The following can be observed: (i) Where no runtime workload execution adaptation takes place, no queries meet their QoS targets expressed in terms of response time. (ii) Queries managed by $Utility_q^{QoS}$ continue to meet (some) stringent QoS targets where the other methods fail – this is because optimization selectively discriminates against some queries where this is necessary to enable others to meet their targets. (iii) Queries managed by $Utility_q^{RT}$ meet more QoS targets than the action based strategies because the optimizer considers the combined costs or benefits of collections of adaptations in a way that is not considered by the action-based approaches. A broader and more detailed description of the approaches and associated experiments is provided by Paton *et al.* [15]. For the purposes of this paper, we note that optimization based on a utility function that aims to maximize the number of QoS targets met has been shown to out-perform action-based strategies and utility based strategies that target different goals, thus illustrating how utility based techniques can target application requirements.

5 Conclusion

This paper presents a utility-based approach for adaptive workload execution, and has illustrated its application to workloads consisting of workflows or queries. Recent research that explicitly focuses on data intensive cloud computing has addressed issues such as evaluation primitives (e.g. [14]) or the development of layered architectures (e.g. [4]). However, results from many different parts of the database community may usefully be revisited in the context of clouds; this paper considers workload management [11], and in particular the use of utility functions for coordinating workload execution. In this setting, a utility-based approach has been shown to be applicable to different types of workload, and utility-based techniques can be applied both to coordinate adaptations at different granularities and to address context-specific optimization goals. These context-specific goals allow utility functions to direct system behavior in a way that reflects the requirements of the contracts or SLAs that are likely to be prominent in cloud computing.

References

[1] R. Agrawal et al. The claremont report on database research. *ACM SIGMOD Record*, 37(3):9–19, 2008.

[2] C. Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. on Optimization*, 17(1):188–217, 2006.

[3] M.N. Bennani and D.A. Menasce. Resource allocation for autonomic data centres using analytic performance models. In *Proc. 2nd ICAC*, pages 229–240. IEEE Press, 2005.

[4] M. Brantner, D. Florescu, D. A. Graf, D. Kossmann, and T. Kraska. Building a database on s3. In *SIGMOD Conference*, pages 251–264, 2008.

[5] S. Chaudhuri, V.R. Narasayya, and R. Ramamurthy. Estimating Progress of Long Running SQL Queries. In *Proc. SIGMOD*, pages 803–814, 2004.

[6] R. Fletcher. *Practical Methods of Optimization*. John Wiley&Sons, 1987.

[7] A. Gounaris, N.W. Paton, A.A.A. Fernandes, and R. Sakellariou. Self-monitoring query execution for adaptive query processing. *Data Knowl. Eng.*, 51(3):325–348, 2004.

[8] J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[9] J.O. Kephart and R. Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[10] S. Krompass, U. Dayal, H. A. Kuno, and A. Kemper. Dynamic workload management for very large data warehouses: Juggling feathers and bowling balls. In *VLDB*, pages 1105–1115, 2007.

[11] S. Krompass, A. Scholz, M.-Cezara Albutiu, H. A. Kuno, J. L. Wiener, U. Dayal, and A. Kemper. Quality of service-enabled management of database workloads. *IEEE Data Eng. Bull.*, 31(1):20–27, 2008.

[12] K. Lee, N.W. Paton, R. Sakellariou, and A.A.A. Fernandes. Utility Driven Adaptive Workflow Execution. In *Proc. 9th CCGrid*. IEEE Press, 2009.

[13] K. Lee, R. Sakellariou, N.W. Paton, and A.A.A. Fernandes. Workflow Adaptation as an Autonomic Computing Problem. In *Proc. 2nd Workshop on Workflows in Support of Large-Scale Science (WORKS 07), Proc. of HPDC & Co-Located Workshops*, pages 29–34. ACM Press, 2007.

[14] H. Liu and D. Orban. Gridbatch: Cloud computing for large-scale data-intensive batch applications. In *CCGRID*, pages 295–305. IEEE Computer Society, 2008.

[15] N.W. Paton, Marcelo A. T. de Aragão, and A.A.A. Fernandes. Utility-driven adaptive query workload execution. In *Submitted for Publication*, 2009.

[16] R. Sakellariou and V. Yarmolenko. Job scheduling on the grid: Towards sla-based scheduling. In L. Grandinetti, editor, *High Performance Computing and Grids in Action*, pages 207–222. IOS, 2008.

[17] M.A. Shah, J.M. Hellerstein, S.Chandrasekaran, and M.J. Franklin. Flux: An adaptive partitioning operator for continuous query systems. In *Proc. ICDE*, pages 353–364. IEEE Press, 2003.

[18] P. Shivam, S. Babu, and J. S. Chase. Active and accelerated learning of cost models for optimizing scientific applications. In *VLDB*, pages 535–546, 2006.

[19] W.E. Walsh, G. Tesauro, J.O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proc. ICAC*, pages 70–77. IEEE Press, 2004.

[20] P. Watson, P. Lord, F. Gibson, P. Periorellis, and G. Pitsilis. Cloud Computing for e-Science with CARMEN. In *2nd Iberian Grid Infrastructure Conference Proceedings*, pages 3–14, 2008.

[21] M. Wieczorek, A. Hoheisel, and P. Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems*, 25(3):237–256, 2009.

Implementation Issues of A Cloud Computing Platform

Bo Peng, Bin Cui and Xiaoming Li
Department of Computer Science and Technology, Peking University
{pb,bin.cui,lxm}@pku.edu.cn

Abstract

Cloud computing is Internet based system development in which large scalable computing resources are provided "as a service" over the Internet to users. The concept of cloud computing incorporates web infrastructure, software as a service (SaaS), Web 2.0 and other emerging technologies, and has attracted more and more attention from industry and research community. In this paper, we describe our experience and lessons learnt in construction of a cloud computing platform. Specifically, we design a GFS compatible file system with variable chunk size to facilitate massive data processing, and introduce some implementation enhancement on MapReduce to improve the system throughput. We also discuss some practical issues for system implementation. In association of the China web archive (Web InfoMall) which we have been accumulating since 2001 (now it contains over three billion Chinese web pages), this paper presents our attempt to implement a platform for a domain specific cloud computing service, with large scale web text mining as targeted application. And hopefully researchers besides our selves will benefit from the cloud when it is ready.

1 Introduction

As more facets of work and personal life move online and the Internet becomes a platform for virtual human society, a new paradigm of large-scale distributed computing has emerged. Web-based companies, such as Google and Amazon, have built web infrastructure to deal with the internet-scale data storage and computation. If we consider such infrastructure as a "virtual computer", it demonstrates a possibility of new computing model, i.e., centralize the data and computation on the "super computer" with unprecedented storage and computing capability, which can be viewed as a simplest form of cloud computing.

More generally, the concept of cloud computing can incorporate various computer technologies, including web infrastructure, Web 2.0 and many other emerging technologies. People may have different perspectives from different views. For example, from the view of end-user, the cloud computing service moves the application software and operation system from desktops to the cloud side, which makes users be able to plug-in anytime from anywhere and utilize large scale storage and computing resources. On the other hand, the cloud computing service provider may focus on how to distribute and schedule the computer resources. Nevertheless, the storage and computing on massive data are the key technologies for a cloud computing infrastructure.

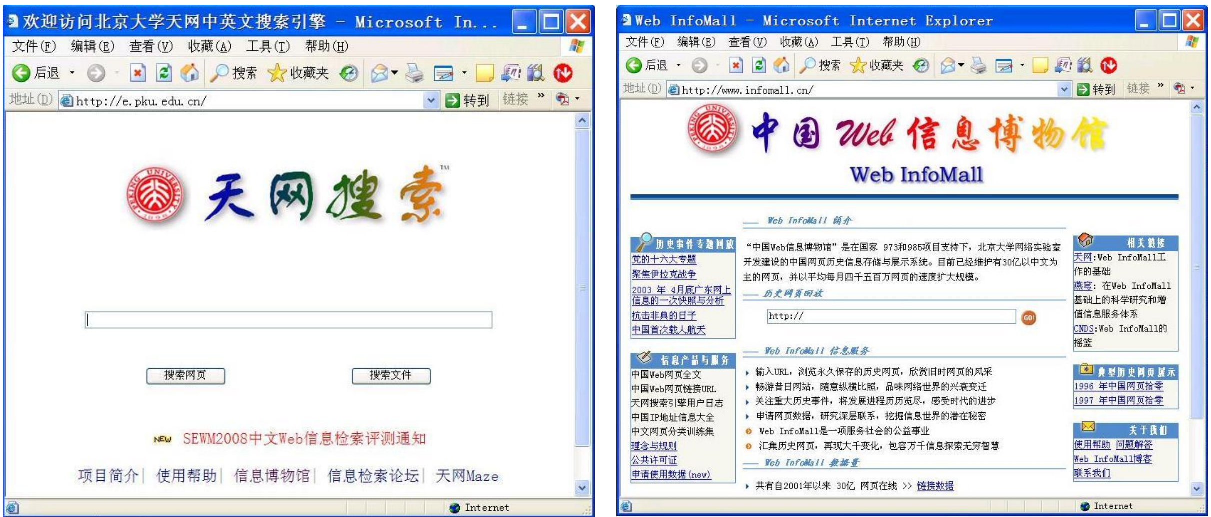
Google has developed its infrastructure technologies for cloud computing in recent years, including Google File System (GFS) [8], MapReduce [7] and Bigtable [6]. GFS is a scalable distributed file system, which

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

emphasizes fault tolerance since it is designed to run on economically scalable but inevitably unreliable (due to its sheer scale) commodity hardware, and delivers high performance service to a large number of clients. Bigtable is a distributed storage system based on GFS for structured data management. It provides a huge three-dimensional mapping abstraction to applications, and has been successfully deployed in many Google products. MapReduce is a programming model with associated implementation for massive data processing. MapReduce provides an abstraction by defining a “mapper” and a “reducer”. The “mapper” is applied to every input key/value pair to generate an arbitrary number of intermediate key/value pairs. The “reducer” is applied to all values associated with the same intermediate key to generate output key/value pairs. MapReduce is an easy-to-use programming model, and has sufficient expression capability to support many real world algorithms and tasks. The MapReduce system can partition the input data, schedule the execution of program across a set of machines, handle machine failures, and manage the inter-machine communication.

More recently, many similar systems have been developed. KosmosFS [3] is an open source GFS-Like system, which supports strict POSIX interface. Hadoop [2] is an active Java open source project. With the support from Yahoo, Hadoop has achieved great progress in these two years. It has been deployed in a large system with 4,000 nodes and used in many large scale data processing tasks.

In Oct 2007, Google and IBM launched “cloud computing initiative” programs for universities to promote the related teaching and research work on increasingly popular large-scale computing. Later in July 2008, HP, Intel and Yahoo launched a similar initiative to promote and develop cloud computing research and education. Such cloud computing projects can not only improve the parallel computing education, but also promote the research work such as Internet-scale data management, processing and scientific computation. Inspired by this trend and motivated by a need to upgrade our existing work, we have implemented a practical web infrastructure as cloud computing platform, which can be used to store large scale web data and provide high performance processing capability. In the last decade, our research and system development focus is on Web search and Web Mining, and we have developed and maintained two public web systems, i.e., *Tianwang* Search Engine [4] and Web Archive system *Web infomall* [1] as shown in Figure 1.



(a) Tianwang (b) Web infomall

Figure 1: Search engine and Chines web archive developed at SEWM group of PKU

During this period, we have accumulated more than 50 TB web data, built a PC cluster consisting of 100+ PCs, and designed various web application softwares such as webpage text analysis and processing. With the increase of data size and computation workload in the system, we found the cloud computing technology is a promising approach to improve the scalability and productivity of the system for web services. Since 2007, we started to design and develop our web infrastructure system, named "Tplatform", including GFS-like file system "TFS" [10] and MapReduce computing environment. We believe our practice of cloud computing platform implementation could be a good reference for researchers or engineers who are interested in this area.

2 TPlatform: A Cloud Computing Platform

In this section, we briefly introduce the implementation and components of our cloud computing platform, named "Tplatform". We first present the overview of the system, followed by the detailed system implementation and some practical issues.

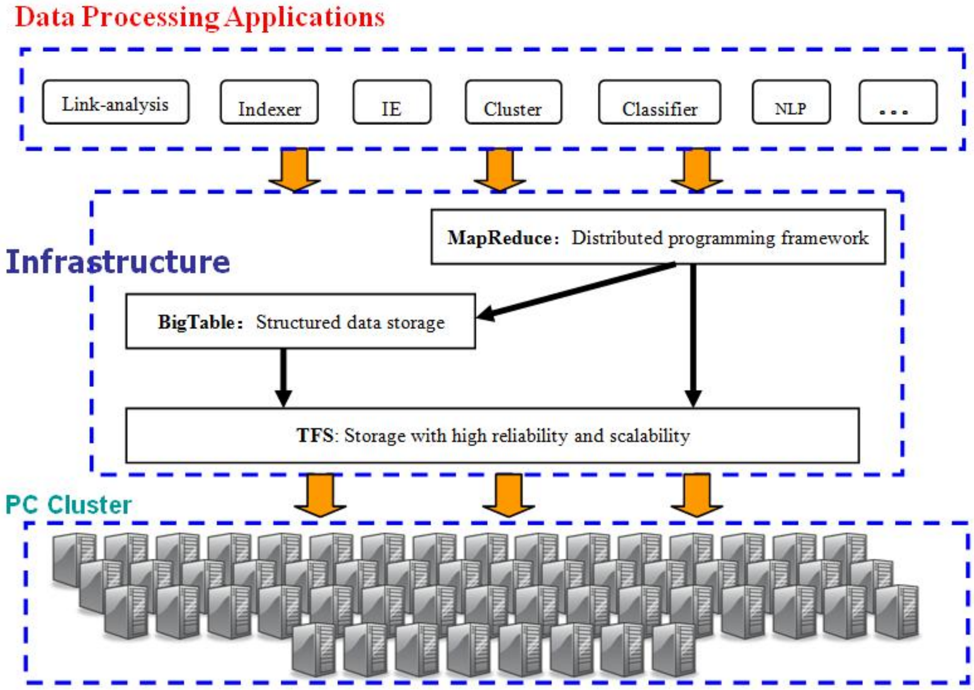


Figure 2: The System Framework of Tplatform

Fig 2 shows the overall system framework of the "Tplatform", which consists of three layers, i.e., PC cluster, infrastructure for cloud computing platform, and data processing application layer. The PC cluster layer provides the hardware and storage devices for large scale data processing. The application layer provides the services to users, where users can develop their own applications, such as Web data analysis, language processing, cluster and classification, etc. The second layer is the main focus of our work, consisting of file system TFS, distributed data storage mechanism BigTable, and MapReduce programming model. The implementation of BigTable is similar to the approach presented in [6], and hence we omit detailed discussion here.

2.1 Implementation of File System

The file system is the key component of the system to support massive data storage and management. The designed TFS is a scalable, distributed file system, and each TFS cluster consists of a single master and multiple chunk servers and can be accessed by multiple client.

2.1.1 TFS Architecture

In TFS, files are divided into variable-size chunks. Each chunk is identified by an immutable and globally unique 64 bit chunk handle assigned by the master at the time of chunk creation. Chunk servers store the chunks on the local disks and read/write chunk data specified by a chunk handle and byte range. For the data reliability, each chunk is replicated on multiple chunk servers. By default, we maintain three replicas in the system, though users can designate different replication levels for different files.

The master maintains the metadata of file system, which includes the namespace, access control information, the mapping from files to chunks, and the current locations of chunks. It also controls system-wide activities such as garbage collection of orphaned chunks, and chunk migration between chunk servers. Each chunk server periodically communicates with the master in HeartBeat messages to report its state and retrieve the instructions.

TFS client module is associated with each application by integrating the file system API, which can communicate with the master and chunkservers to read or write data on behalf of the application. Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunkservers.

The system is designed to minimize the master's involvement in file accessing operations. We do not provide the POSIX API. Besides providing the ordinary read and write operations, like GFS, we have also provided an atomic record appending operation so that multiple clients can append concurrently to a file without extra synchronization among them. In the system implementation, we observe that the record appending operation is the key operation for system performance. We design our own system interaction mechanism which is different from GFS and yields better record appending performance.

2.1.2 Variable Chunk Size

In GFS, a file is divided into fixed-size chunks (e.g., 64 MB). When a client uses record appending operation to append data, the system checks whether appending the record to the last chunk of a certain file may make the chunk overflowed, i.e., exceed the maximum size. If so, it pads all the replica of the chunk to the maximum size, and informs the client that the operation should be continued on the new chunk. (Record appending is restricted to be at most one-fourth of the chunk size to keep worst case fragmentation at an acceptable level.) In case of write failure, this approach may lead to duplicated records and incomplete records.

In our TFS design, the chunks of a file are allowed to have variable sizes. With the proposed system interaction mechanism, this strategy makes the record appending operation more efficient. Padding data, record fragments and record duplications are not necessary in our system. Although this approach brings some extra cost, e.g., every data structure of chunk needs a chunk size attribute, the overall performance is significantly improved, as the read and record appending operations are the dominating operations in our system and can benefit from this design choice.

2.1.3 File Operations

We have designed different file operations for TFS, such as read, record append and write. Since we allow variable chunk size in TFS, the operation strategy is different from that of GFS. Here we present the detailed implementation of read operation to show the difference of our approach.

To read a file, the client exchanges messages with the master, gets the locations of chunks it wants to read from, and then communicates with the chunk servers to retrieve the data. Since GFS uses the fixed chunk size, the client just needs to translate the file name and byte offset into a chunk index within the file, and sends the master a request containing the file name and chunk index. The master replies with the corresponding chunk handle and locations of the replicas. The client then sends a request to one of the replicas, most likely the closest one. The request specifies the chunk handle and a byte range within that chunk. Further reads of the same chunk do not require any more client-master interaction unless the cached information expires or the file is reopened.

In our TFS system, the story is different due to the variable chunk size strategy. The client can not translate the byte offset into a chunk index directly. It has to know all the sizes of chunks in the file before deciding which chunk should be read. Our solution is quite straightforward, when a client opens a file using read mode, it gets all the chunks' information from the master, including chunk handle, chunk size and locations, and use these information to get the proper chunk. Although this strategy is determined by the fact of variable chunk size, its advantage is that the client only needs to communicate with the master once to read the whole file, which is much efficient than GFS' original design. The disadvantage is that when a client has opened a file for reading, later appended data by other clients is invisible to this client. But we believe this problem is negligible, as the majority of the files in web applications are typically created and appended once, and read by data processing applications many times without modifications. If in any situation this problem becomes critical, it can be easily overcome by set an expired timestamp for the chunks' information and refresh it when invalid.

The TFS demonstrates our effort to build an infrastructure for large scale data processing. Although our system has the similar assumptions and architectures as GFS, the key difference is that the chunk size is variable, which makes our system able to adopt different system interactions for record appending operation. Our record appending operation is based on chunk level, thus the aggregate record appending performance is no longer restricted by the network bandwidth of the chunk servers that store the last chunk of the file. Our experimental evaluation shows that our approach significantly improves the concurrent record appending performance for single file by 25%. More results on TFS have been reported in [10]. We believe the design can apply to other similar data processing infrastructures.

2.2 Implementation of MapReduce

MapReduce system is another major component of the cloud computing platform, and has attracted more and more attentions recently [9, 7, 11]. The architecture of our implementation is similar to Hadoop [2], which is a typical master-worker structure. There are three roles in the system: Master, Worker and User. Master is the central controller of the system, which is in charge of data partitioning, task scheduling, load balancing and fault tolerance processing. Worker runs the concrete tasks of data processing and computation. There exist many workers in the system, which fetch the tasks from Master, execute the tasks and communicate with each other for data transfer. User is the client of the system, implements the Map and Reduce functions for computation task, and controls the flow of computation.

2.2.1 Implementation Enhancement

We make three enhancements to improve the MapReduce performance in our system. First, we treat intermediate data transfer as an independent task. Every computation task includes map and reduce subtasks. In a typical implementation such as Hadoop, reduce task starts the intermediate data transfer, which fetches the data from all the machines conducting map tasks. This is an uncontrollable all-to-all communication, which may incur network congestion, and hence degrade the system performance. In our design, we split the transfer task from the reduce task, and propose a "Data transfer module" to execute and schedule the data transfer task independently. With appropriate scheduling algorithm, this method can reduce the probability of network congestion. Although this approach may aggravate the workload of Master when the number of transfer tasks is large, this problem can be alleviated by adjusting the granularity of transfer task and integrating data transfer tasks with the same source and target addresses. In practice, our new approach can significantly improve the data transfer performance.

Second, task scheduling is another concern on MapReduce system, which helps to commit resources between a variety of tasks and schedule the order of task execution. To optimize the system resource utility, we adopt multi-level feedback queue scheduling algorithm in our design. Multiple queues are used to allocate the concurrent tasks, and each of them is assigned with a certain priority, which may vary for different tasks with respect to the resources requested. Our algorithm can dynamically adjust the priority of running task, which balances the system workload and improves the overall throughput.

The third improvement is on data serialization. In MapReduce framework, a computation task consists of four steps: map, partition, group and reduce. The data is read in by map operation, intermediate data is generated and transferred in the system, and finally the results are exported by reduce operation. There exist frequent data exchanges between memory and disk which are generally accomplished by data serialization. In our implementation of MapReduce system, we observed that the simple native data type is frequently used in many data processing applications. Since memory buffer is widely used, most of the data already reside in the memory before they are de-serialized into a new data object. In other words, we should avoid expensive de-serialization operations which consume large volume of memory space and degrade the system performance. To alleviate this problem, we define the data type for key and value as void* pointer. If we want to de-serialize the data with native data type, a simple pointer assignment operation can replace the de-serialization operation, which is much more efficient. With this optimization, we can also sort the data directly in the memory without data de-serialization. This mechanism can significantly improve the MapReduce performance, although it introduces some cost overhead for buffer management.

2.2.2 Performance Evaluation on MapReduce

Due to the lack of benchmark which can represent the typical applications, performance evaluation on MapReduce system is not a trivial task. We first use PennySort as the simple benchmark. The result shows that the performance of intermediate data transfer in the shuffle phase is the bottle neck of the system, which actually motivated us to optimize the data transfer module in MapReduce. Furthermore, we also explore a real application for text mining, which gathers statistics of Chinese word frequency in webpages. We run the program on a 200GB Chinese Web collection. Map function analyzes the content of web page, and produces every individual Chinese word as the key value. Reduce function sums up all aggregated values and exports the frequencies. In our testbed with 18 nodes, the job was split into 3385 map tasks, 30 reduce tasks and 101550 data transfer tasks, the whole job was successfully completed in about 10 hours, which is very efficient.

2.3 Practical Issues for System Implementation

The data storage and computation capability are the major factors of the cloud computing platform, which determine how well the infrastructure can provide services to end users. We met some engineering and technical problems during the system implementation. Here we discuss some practical issues in our work.

2.3.1 System Design Criteria

In the system design, our purpose is to develop a system which is scalable, robust, high-performance and easy to be maintained. However, some system design issues may be conflicted, which places us in a dilemma in many cases. Generally, we take three major criteria into consideration for system design: 1) For a certain solution, what is bottleneck of the procedure which may degenerate the system performance? 2) Which solution has better scalability and flexibility for future change? 3) Since network bandwidth is the scarce resource of the system, how to fully utilize the network resource in the implementation? In the following, we present an example to show our considerations in the implementation.

In the MapReduce system, fault tolerance can be conducted by either master or workers. Master takes the role of global controller, maintains the information of the whole system and can easily decide whether a failed task should be rerun, and when/where to be rerun. Workers only keep local information, and take charge of reporting the status of running tasks to Master. Our design combines the advantages of these two factors. The workers can rerun a failed task for a certain number of times, and are even allowed to skip some bad data records which cause the failure. This distributed strategy is more robust and scalable than centralized mechanism, i.e., only re-schedule failed tasks in the Master side.

2.3.2 Implementation of Inter-machine Communication

Since the implementation of cloud computing platform is based on the PC cluster, how to design the inter-machine communication protocol is the key issue of programming in the distributed environment. The Remote Procedure Call (RPC) middle ware is a popular paradigm for implementing the client-server model of distributed computing, which is an inter-process communication technology that allows a computer program to cause a subroutine or procedure to execute on another computer in a PC cluster without the programmer explicitly coding the details for this remote interaction. In our system, all the services and heart-beat protocols are RPC calls. We exploit Internet Communications Engine (ICE), which is an object-oriented middleware that provides object-oriented RPC, to implement the RPC framework. Our approach performs very well under our system scale and can support asynchronous communication model. The network communication performance of our system with ICE is comparable to that of special asynchronous protocols with socket programming, which is much more complicated for implementation.

2.3.3 System Debug and Diagnosis

Debug and Diagnosis in distributed environment is a big challenge for researchers and engineers. The overall system consists of various processes distributed in network, and these processes communicate each other to execute a complex task. Because of the concurrent communications in such system, many faults are generally not easy to be located, and hence can hardly be debugged. Therefore, we record complete system log in our system. In All the server and client sides, important software boundaries such as API and RPC interfaces are all logged. For example, log for RPC messages can be used to check integrality of protocol, log for data transfer can be used to validate the correctness of transfer. In addition, we record performance log for performance tuning. In our MapReduce system, log in client side records the details of data read-in time, write-out time of all tasks, time cost of sorting operation in reduce task, which are tuning factors of our system design.

In our work, the recorded log not only helps us diagnose the problems in the programs, but also helps find the performance bottleneck of the system, and hence we can improve system implementation accordingly. However, distributed debug and diagnosis are still low efficient and labor consuming. We expect better tools and approaches to improve the effectiveness and efficiency of debug and diagnosis in large scale distributed system implementation.

3 Conclusion

Based on our experience with Tplatform, we have discussed several practical issues in the implementation of a cloud computing platform following Google model. It is observed that while GFS/MapReduce/BigTable provides a great conceptual framework for the software core of a cloud and Hadoop stands for the most popular open source implementation, there are still many interesting implementation issues worth to explore. Three are identified in this paper.

- The chunksize of a file in GFS can be variable instead of fixed. With careful implementation, this design decision delivers better performance for read and append operations.
- The data transfer among participatory nodes in reduce stage can be made ”schedulable” instead of ”un-controlled”. The new mechanism provides opportunity for avoiding network congestions that degrade performance.
- Data with native types can also be effectively serialized for data access in map and reduce functions, which presumably improves performance in some cases.

While Tplatform as a whole is still in progress, namely the implementation of BigTable is on going, the finished parts (TFS and MapReduce) are already useful. Several applications have shown the feasibility and advantages of our new implementation approaches. The source code of Tplatform is available from [5].

Acknowledgment

This work was Supported by 973 Project No. 2007CB310902, IBM 2008 SUR Grant for PKU, and National Natural Science foundation of China under Grant No.60603045 and 60873063.

References

[1] *China Web InfoMall*. http://www.infomall.cn, 2008.

[2] *The Hadoop Project*. http://hadoop.apache.org/, 2008.

[3] *The KosmosFS Project*. http://kosmosfs.sourceforge.net/, 2008.

[4] *Tianwang Search*. http://e.pku.edu.cn, 2008.

[5] *Source Code of Tplatform Implementation*. http://net.pku.edu.cn/˜ webg/tplatform, 2009.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 15–15, 2006.

[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI '04: Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, pages 137–150, 2004.

[8] G. Sanjay, G. Howard, and L. Shun-Tak. The google file system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 29–43, 2003.

[9] H. Yang, A. Dasdan, R. Hsiao, and D. S. Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040, 2007.

[10] Z. Yang, Q. Tu, K. Fan, L. Zhu, R. Chen, and B. Peng. Performance gain with variable chunk size in gfs-like file systems. In *Journal of Computational Information Systems*, pages 1077–1084, 2008.

[11] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI '07: Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, pages 29–42, 2007.

Dataflow Processing and Optimization on Grid and Cloud Infrastructures*

M. Tsangaris, G. Kakaletris, H. Kllapi, G. Papanikos, F. Pentaris,
P. Polydoras, E. Sitaridi, V. Stoumpos, Y. Ioannidis
Dept. of Informatics & Telecom, MaD*g*IK Lab, University of Athens, Hellas (Greece)
{mmt,gkakas,herald,g.papanikos,frank,p.polydoras,evas,stoumpos,yannis}@di.uoa.gr
http://madgik.di.uoa.gr/

Abstract

Complex on-demand data retrieval and processing is a characteristic of several applications and combines the notions of querying & search, information filtering & retrieval, data transformation & analysis, and other data manipulations. Such rich tasks are typically represented by data processing graphs, having arbitrary data operators as nodes and their producer-consumer interactions as edges. Optimizing and executing such graphs on top of distributed architectures is critical for the success of the corresponding applications and presents several algorithmic and systemic challenges. This paper describes a system under development that offers such functionality on top of Ad-hoc Clusters, Grids, or Clouds. Operators may be user defined, so their algebraic and other properties as well as those of the data they produce are specified in associated profiles. Optimization is based on these profiles, must satisfy a variety of objectives and constraints, and takes into account the particular characteristics of the underlying architecture, mapping high-level dataflow semantics to flexible runtime structures. The paper highlights the key components of the system and outlines the major directions of its development.

1 Introduction

Imagine you have developed an innovative web-based search service that you would like to offer to the world. Cloud Computing enables you to host this service remotely and deal with scale variability: as your business grows or shrinks, you can acquire or release Cloud resources easily and relatively inexpensively. On the other hand, implementation and maintenance of data services that are scalable and adaptable to such dynamic conditions becomes a challenge. This is especially the case for data services that are compositions of other, possibly third-party services (e.g., Google Search or Yahoo Image Search), where the former become data processing graphs that use the latter as building blocks (nodes) and invoke them during their execution. Running services under various quality-of-service (QoS) constraints that different customers may desire adds further complications. Handcrafting data processing graphs that implement such services correctly, make optimal use of the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Partially supported by the European Commission under contracts INFRA-2007-212488, FP6-IST-027749, and FP6-IST-004260, for the D4Science, Health-e-Child, and DILIGENT projects, respectively.

resources available, and satisfy all QoS and other constraints is a daunting task. Automatic dataflow optimization and execution are critical for data services to be scalable and adaptable to the Cloud environment.

This is in analogy to query optimization and execution in traditional databases but with the following differences: component services may represent arbitrary operations on data with unknown semantics, algebraic properties, and performance characteristics, and are not restricted to come from a well-known fixed set of operators (e.g., those of relational algebra); optimality may be subject to QoS or other constraints and may be based on multiple diverse relevant criteria, e.g., monetary cost of resources, staleness of data, etc., and not just solely on performance; the resources available for the execution of a data processing graph are flexible and reservable on demand and are not fixed a-priori. These differences make dataflow optimization essentially a new challenging problem; they also generate the need for run-time mechanisms that are not usually available.

This paper presents **ADP** (Athena Distributed Processing), a distributed dataflow processing system that attempts to address the above challenges on top of Ad-Hoc Clusters (physical computer nodes connected with a fast local or wide-area network), Grids [5], and Clouds [9], each time adapting itself to the particular characteristics or constraints of the corresponding architecture. These architectures do not represent arbitrary unrelated choices, but can be considered as distinct points in an evolutionary path. While an Ad-Hoc cluster simply provides raw compute power, the Grid additionally provides mechanisms for managing computational, storage, and other resources in a synergistic way. When evolving from Grids to Clouds, additional resources are made available for lease, offering opportunities for more complex systemic scenarios, but also making service scalability, and performance, and composability even more challenging.

The paper begins with the internal representations of ADP queries. It continues with the runtime system of ADP, the stand-alone representations of operator properties, and the key features of its query optimization. It concludes with the implementation status of ADP, a comparison with related work, and some future directions.

2 ADP Query Language Abstractions

User requests to ADP take the form of queries in some high-level declarative or visual language, not described here. Internally, they are represented by equivalent queries in procedural languages at various abstraction levels:

Operator Graphs: These are the queries in **ADFL** (Athena Data Flow Language), the main internal ADP language. Their nodes are data **operators** and their (directed) edges are operator interactions in the form of producing and consuming **dataflows** (or simply **flows**). Operators encapsulate data processing algorithms and may be custom-made by end users. Flows originate from operators, are transformed by operators, and are delivered as results by the root operator of a query. A flow is a *finite sequence* of **records**. ADP treats records as abstract data containers during processing. Their properties (e.g., type name, type compatibility, keys, size) are stored in **record profiles** and play an important role when establishing operator-to-(operator or end-user) flows.

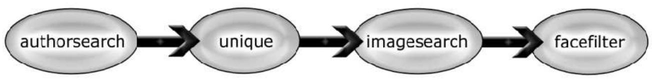


Figure 1: The Query Operator Graph

Example: Suppose a user wants to search the Web for images of authors of IEEE and ACM papers. Figure 1 shows an ADFL query that corresponds to this need. The query is essentially a chain (composition) of operators: First, there is a custom operator, AUTHORSEARCH, communicating with some particular external digital-library service to retrieve author names, filtered to select only authors of IEEE and ACM papers. Then, the standard operator UNIQUE eliminates duplicate author names. Next, the resulting flow of author names is sent to another custom operator, IMAGESEARCH, which uses the names to identify corresponding images in an external database. Finally, a face detection operator, FACEFILTER, is used to identify images that contain only faces and forced to output only the best match (topk=”1”). In text form, the query in Figure 1 is expressed as

FACEFILTER{topk=”1”} ON IMAGESEARCH ON UNIQUE ON AUTHORSEARCH{pub=”IEEE” or ”ACM”} □

Concrete Operator Graphs: These are similar to operator graphs but their nodes are **concrete operators**, i.e., software components that implement operators in a particular way and carry all necessary details for their execution. The UNIQUE operator, for example, has to store all records (or record keys) seen so far; it may be realized as two alternative Java implementations, one based on main memory (fast but limited by memory size) and one based on external storage (slower but limited only by disk size). Choosing among them becomes an optimization decision, based for example, on expected input flow sizes.

Subject to its initialization, as part of its execution, a concrete operator may be contacting external services to retrieve data from them. In that case, it must deal with all physical, security, and semantic issues related to such external communication. This is transparent, however, to the operator that consumes the flow resulting from such external services: all sources of flows appear the same, independent of any external interactions.

Execution Plans: These are similar to concrete operator graphs, but their nodes are concrete operators that have been allocated resources for execution and have all their initialization parameters set.

3 ADP Runtime Environment

Concrete operator graph queries are eventually evaluated by the **ART** (ADP Run Time) subsystem, which has two main software parts: **Containers** are responsible for supervising concrete operators and providing the necessary execution context for them (memory resources, security credentials, communication mechanisms, etc.). **ResultSets** are point-to-point links to transport polymorphic records between concrete operators and implement the query flows. While different manifestations of data are supported, e.g., native objects, byte-streams, or XML content, ResultSets are type agnostic.

Containers are the units of resource allocation between ART and the processing nodes of the underlying distributed infrastructure. Based on the optimizer’s decisions, ART dynamically creates or destroys containers to reflect changes in the system load. Thus, a complex query may be distributed across multiple containers running on different computer systems, each container being responsible for one or more of the query’s concrete operators and ResultSets. Likewise, based on the optimizer’s decisions, ResultSets control the mode of record transportation, ranging from pipelining (totally synchronous producer/consumer operation) to store & forward (full buffering of entire flow before making it available to the consumer).

Containers feature the same set of tools to support ADFL query evaluation but are implemented differently depending on the characteristics of the underlying distributed infrastructure architecture, hiding all lower-level architectural details and the corresponding technology diversity. Containers on Ad-Hoc clusters or Clouds, for example, may simply be just processes, while on Grid, they may be Web Service containers. ResultSets may utilize WebService transport (SOAP) on Grids, or simply TCP sockets on Ad-Hoc Clusters or Clouds. The optimizer may try to minimize Cloud resource lease cost by shutting down some or not using several containers, while this is not an issue in other architectures.

The runtime mechanisms provided for query execution have a major impact on application development in that they liberate implementers from dealing with execution platform or data communication details. Note that there is a particularly good match between Cloud architectures and certain characteristics of ADP: Custom operators within ADFL queries are an easy and attractive way to use ad hoc third-party services (e.g., AUTHORSEARCH or FACEFILTER), which is an important feature of Clouds. More importantly, dynamic acquisition and release of resources (containers and virtual machines) by ADP as a systemic response to load or QoS requirements fits very well with the canonical Cloud business model; the presence of Service Level Agreements (SLAs) that must be met requires such flexible resource allocation, which in turn, calls for sophisticated optimization of the kind ADP is designed to offer. The need for advanced optimization strategies is less marked in other architectures, e.g., in Grid, where simple matching of operations to resources usually suffices.

4 Operator Profiles

Given the ad hoc nature of most ADP operators, no pertinent information about them is hardwired into the system; instead it is all provided by users and stored in **operator profiles**. Typically, for each level of internal language abstraction, there is relevant information in an operator's profile. Accordingly, ADP uses the profile contents recursively to drive the corresponding stages of query optimization and execution. Below, we indicate some fundamental properties that may be found in (or derived from) an operator's profile for each abstraction level, emphasizing primarily those that generate equivalent alternatives that the optimizer must examine when a query with that operator is considered. We avoid describing the precise structure/schema of the profile or the language used to express some of its contents; instead, we use a stylized pseudo-language for easy exposition.

Operator Graphs: At this level, in addition to its signature (input/output flows with specific record profiles), of great importance are algebraic equivalences that operators satisfy. These include typical **algebraic transformations**, e.g., associativity, commutativity, or distributivity, **(de)compositions**, i.e., operators being abstractions of whole operator graphs that involve compositions, aggregations, and other interactions of more specific operators, and **partitions**, i.e., operators being amenable to replication and parallel processing by each replica of part of the original input, in conjunction with some pre- and post-processing operators.

Example: Consider the following information being known about the operators of Figure 1:

- Algebraic transformation - Filtered on multiple publishers, AUTHORSEARCH is equivalent to merging the results of itself filtered on each one of them separately:
  operator AUTHORSEARCH{pub=x or y} is MERGE on AUTHORSEARCH{pub=x} and AUTHORSEARCH{pub=y};
- Operator decomposition - Filtered on IEEE or ACM, AUTHORSEARCH is equivalent to another operator that searches directly the IEEE or ACM Digital Libraries, respectively:
  operator AUTHORSEARCH{pub="IEEE"} is IEEESEARCH;
  operator AUTHORSEARCH{pub="ACM"} is ACMSEARCH;
- Operator partition - FACEFILTER is trivially parallelizable on its input, with operators SPLIT and MERGE performing the necessary flow pre- and post-processing, before and after the parallel execution of an arbitrary (unspecified) number of FACEFILTER instances:
  operator FACEFILTER is splitable with {pre-process = SPLIT ; post-process = MERGE}; □

Concrete Operator Graphs: At this level, capturing an operator's available implementation(s) is the critical information. In general, there may be multiple concrete operators implementing an operator, e.g., a low-memory but expensive version and a high-memory but fast one; a multi-threaded version and a single-threaded one; or two totally different but logically equivalent implementations of the same operator. For example, there may be a standard UNIQUE implementation determining record equality based on the entire record, while an alternative custom implementation may only look at a specific key record attribute. Also, IMAGESEARCH may have just a multi-threaded implementation associated with it, but FACEFILTER may have both a single-threaded and a multi-threaded one. All these concrete operators should be recorded in the corresponding operator's profile.

Execution Plan: At this level, the profile of a concrete operator stores information about its multiple potential instantiations in a container, its initialization parameter values, and any constraints on resources it may use, e.g., number of threads, size of memory, software licenses, input/output rates, communication channels, etc. It also stores information about how the optimizer may evaluate a particular instantiation of the concrete operator. For example, the multi-threaded concrete operators for IMAGESEARCH and FACEFILTER have several additional degrees of freedom at the execution plan level, as they can use multiple local CPUs and cores.

5 Query Optimization

Evaluation Parameters: Evaluation of query execution plans is at the heart of query optimization, regarding both the objective function being optimized and any (QoS or other) constraints being satisfied. Depending on the application, such evaluation may be based on a variety of parameters, e.g., monetary cost of resources or freshness of data, and not just solely as is traditional on performance metrics. Given the ad hoc nature of operators, their profiles store mathematical formulas to describe such parameters and any properties of their inputs and outputs that are deemed relevant, e.g., image resolution for FACEFILTER cpu cost, or image database age for IMAGESEARCH freshness. Consequently, for any parameter that may be important to the operators' evalution, statistics should be either maintained or obtained, for example, on the fly through some sampling. Similarly, the mathematical formulas associated with the evaluation of an operator may be either explictly inserted into its profile by some user or predicted based on some sample or prior executions of the operator. ADP is designed to offer generic functionality for synthesizing appropriate formulas and propagating parameter values through the operators of an execution plan to obtain its final evaluation.

Space of Alternatives: Transformation of an ADFL query to an execution plan that can generate the requested results goes through several stages, corresponding to the levels of internal language abstractions, where every alternative in one level has multiple alternative mappings to the next lower level according to the properties in the profiles of the operators involved. There are several operator graphs that are algebraicly equivalent to the original query, each one mapping to several concrete operator graphs (based on the corresponding mappings of its operators), each one mapping to several execution plans by instantiating containers and ResultSets and assigning the instantiated concrete operators and flows of the concrete operator graph to them.

Example: The algebraic properties in the profiles of AUTHORSEARCH and FACEFILTER (assuming 3-way parallelism for the latter) generate the operator graph indicated in Figure 2 as an alternative to Figure 1. Instantiating an execution plan for that graph requires choosing concrete operators and then: container instantiation - the set of containers available to the query are chosen, through dynamic release or acquisition of containers and (virtual) hosts, or reuse of existing ones; concrete operator instantiation - all concrete operators are initialized (e.g., the number of threads for the multi-threaded implementations of IMAGESEARCH and FACEFILTER is set) and assigned to containers; flow instantiation - connected as inputs and outputs of concrete operators, the endpoints of each flow are instantiated via technology-specific endpoint implementations of the ResultSet, fine-tuned for the flow's and connected operators' needs. Figures 2 and 3 indicate particular alternatives with respect to these choices, at the level of the operator graph and the execution plan, respectively. □

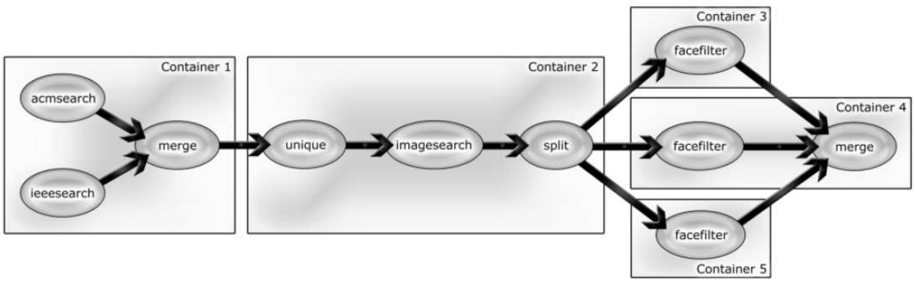


Figure 2: Query Operator Graph after all operator transformations and assignments to containers

Optimization Stages: In principle, optimization could proceed in one giant step, examining all execution plans that could answer the original query and choosing the one that is optimal and satisfies the required constraints. Alternatively, given the size of the alternatives' space, optimization could proceed in multiple smaller steps, each one operating at some level and making assumptions about the levels below. ADP optimization currently proceeds in three distinct steps, corresponding exactly to the three language abstraction levels of ADP.

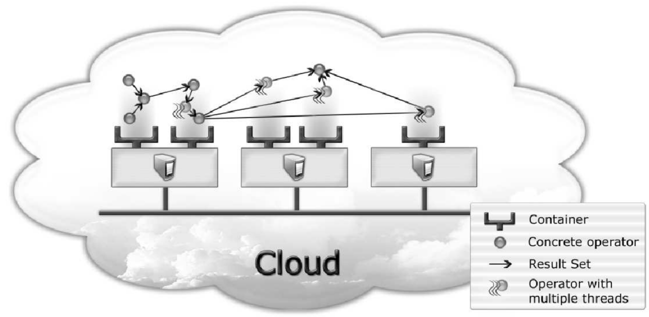


Figure 3: Query Execution Plan with explicit mapping of operators to containers on Cloud (virtual) hosts

The techniques developed for the first two steps are not discussed here due to space limitations.

Execution Plan Instantiation: Assignment of concrete operators to containers is currently modeled and implemented in ADP as a constraint satisfaction problem (CSP) as follows: Consider N containers $L_1, L_2, \ldots, L_N$, each with a current resource capability $CAP(L_i), 1 \leq i \leq N$, and a host $H(L_i)$ where it resides. Let $NETCAP(L_i, L_j)$, $1 \leq i, j \leq N$ be the network capacity (bandwidth) between $H(L_i)$ and $H(L_j)$. Let $P_1, P_2, \ldots, P_M$ be the set of concrete operators present in the respective operator graph and $DEMAND(P_i)$, $1 \leq \text{i} \leq M$, be the resources that operator P_i requires. The CSP solved is to assign each concrete operator to a container ($C(P_i) = L$) so that a user-defined cost function (e.g., $\sum_{1 \leq i,j \leq M} NETCOST(P_i, P_j)$, network cost between operators) is optimized subject to the constraints:

- container resource capability is not exhausted: $\sum_{C(P_i)=L} DEMAND(P_i) \leq CAP(L)$,
- internode bandwidth is not exhausted: $\sum_{C(P_i)=L,\ C(P_j)=L'} NETCOST(P_i, P_j) \leq NETCAP(L, L')$.

The above problem can naturally be expanded to model more complex situations, e.g., taking into account operator gravity (preference for operators to be assigned to the same container), or different optimization objectives.

In high-load conditions, we deploy an admission control algorithm to ensure optimal balancing of workload. Before entering the CSP solution process, the optimizer broadcasts information on the concrete operator graph and "asks" containers to declare which operators they can instantiate (if asked to do so). Containers monitor these requests as well as the actual optimizer decisions and use this information to restrict the number and type of concrete operators they are willing to instantiate. Each container makes potentially a different decision, which are all then used to restrict the space of possible CSP solutions. If the existing containers' decisions do not allow all concreted operators of a query to be instantiated so that an execution plan may be obtained, then the query is automatically resubmitted after a short time expecting greater container availability. If this is not the case, additional containers are requested based on the number and type of concrete operators that are unassigned.

The algorithm used by containers to restrict the amount and type of operators they are willing to instantiate follows the lines of [7]. In critical load situations, it effectivelly diverts resources used by concrete operators that are rarely requested to the ones that are frequently used. This is achieved using the following microeconomics-inspired mechanism: Assuming that C is the number of concrete operators, containers internally hold a private vector $\vec{p} \in \mathbb{R}_+^C$ of virtual concrete operator prices. These prices are never disclosed; they only provide to the admission control algorithm the means to measure the contribution of a concrete operator to the performance of the whole distributed system. As demand for a concrete operator increases/decreases, its respective prices increase/decrease as well. Each container uses its privately held prices to periodically (every t units of time) select a vector $\vec{s} \in \mathbb{N}^C$ of operators to admit. This vector is different for each container, represents the type and number of operators admitted, and is the one that maximizes the virtual price ($\vec{s} \cdot \vec{p}$) of the admitted operators under the resource constraints of each container. That is, each container solves $\max_{\vec{s}} \vec{s} \cdot \vec{p}$ so that $\vec{s}$ is feasible, i.e., the container has enough resources to instantiate all conreate operators in $\vec{s}$ within t units of time.

6 Implementation Status

An initial implementation of ADP is in operation for a year now and is used by data mining and digital library search applications. The ADFL language enables ad-hoc operators to be introduced, without the need to change its parser. Operator profiles contain information such as the java classes implementing operators. ART uses Axis-on-Tomcat Web Service containers, each one running on an Ad-Hoc cluster node or on the Grid. The optimizer performs simple rewriting driven by operator profiles, changes the number of containers dynamically based on current “load”, and decides how to assign concrete operators to containers, as described above. A simulated annealing optimization engine is used to generate the execution plan, based on the concrete operator graph. ResultSets have been implemented, attached to the producing operator, and communicating via TCP sockets or SOAP messages to the consuming operators. Slotted Records model relational database table rows, whereas XML Records provide additional structure. Finally, in addition to the default execution engine, there is a second implementation based on BPEL (Business Process Execution Language), as well as a proof-of-concept “standalone ADP” implementation, which is a single java executable including an ADFL parser, ART, a single container, a library of operator implementations, and a ResultSet implementation.

7 Related Work

ADP provides both a testbed for validating research ideas on distributed data processing and a core platform for supporting several data-intensive distributed infrastructures. It has been influenced by lessons learned from on-going work on data services in the areas of Digital Libraries, e-Health, Earth Sciences, etc., which all need a scalable software layer that can perform compute-intensive data tasks easily, reliably, and with low application complexity. The ADP concepts have been validated in the context of the DILIGENT [12], Health-e-Child [14], and D4Science [13] projects, and form the heart of the gCube system’s information retrieval facilities [15]. Although several core ADP concepts can be found elsewhere as well, integrating them in one system and handling the resulting increased complexity does not appear common. ADP incorporates ideas from databases, streams, distributed processing, and service composition to address the relevant challenges and offer a flexible system that can hide the complexities of its underlying architectural environment.

Typically, some middleware is used to execute user-defined code in distributed environments. In the Grid, OGSA-DAI [2] formalizes access to and exchange of data. Paired up with OGSA-DQP [1], it also addresses query optimisation and scheduling. The Condor / DAGMan / Stork set is a representative technology of the High Performance Computing area. Its capacities for scheduling, monitoring and failure resilience render it a robust and easily scalable mechanism for exploiting vast scientific infrastructures of (mostly) computational resources. Furthermore, Pegasus [4] supports a higher lever of abstraction for both data and operations, and therefore offers true optimization features, as opposed to simple matching of operators to a fixed set of containers.

ADP builds on top of these technologies and introduces ADFL to describe user-defined code in a semantically, technologically and operationally domain agnostic manner. The definition of an Operator, and most importantly the Operator Profile, is by itself a new challenge, since traditional database operations like query re-writing, cost-estimation, completion times, selectivity, co-location capacities/requirements, etc., are not a-priori defined or not known at all. Distributed databases do support custom operations on data via functions or (extended) stored procedures and handle data exchange (i.e., flows) via efficient proprietary mechanisms, but optimization is based on established assumptions of (distributed) relational databases.

The notion of processing multiple dataflows in ADFL is also common in the literature. In its more recent form, Mashups (such as Yahoo! Pipes [16], Google Mashup Editor [10] and Microsoft Popfly [11]) carry out content processing over well known sources (RSS, ATOM, HTTP). The visual languages in these systems serve as a starting point for ADFL, which in addition, deals with alternative representations for Operator Profiles. In (e-)Business integration, workflow languages such as WS-BPEL are used to express complex queries that call for systems that support multiple execution granularities, planning, execution, and monitoring mechanisms, etc. Compared to these systems, ADP is designed to offer a rich query rewriting alternatives in the query optimizer. Finally, SawZall[8] and PigLatin [6] use a higher-lever query language that is executed on MapReduce [3] systems that support massive parallelization and achieve failure resilience. However, the language model of the MapReduce framework is somewhat restricted and restricts opportunities for optimization.

8 Conclusions and On-Going Work

We have given a high-level description of ADP, a distributed dataflow processing system under development, which is designed to run on top of Ad-Hoc Clusters, Grids, and Clouds, in an adaptive manner. It deals with dataflow queries that involve user-defined operators, stores the operators' properties in profiles, and uses those to optimize queries at several levels. Optimization may be based on diverse optimality criteria and constraints but currently focuses on the conventional cpu work parameters.

Work on ADP moves in several directions. These include expressive declarative languages on top of ADFL, mechanisms to deal with operators that preserve state, and fine-grade security. On the optimization side, the focus is on Cloud-related architectures, on refining dynamic resource acquisition and release, and on dealing with complex constraints on these resources. Additionally, the role of execution risk in ADP operators is being ivestigated, in scenarios where different plans are exposed to different execution risk profiles and users have different attitudes towards risk (e.g., risk aversion).

References

[1] M. N. Alpdemir, A. Mukherjee, N. W. Paton, P. Watson, A. A. A. Fernandes, A. Gounaris, and J. Smith, "Service-based Distributed Querying on the Grid," in *ICSOC*, 2003, pp. 467–482.

[2] M. Atkinson et al., "A new Architecture for OGSA-DAI," in *Proceedings of the UK e-Science All Hands Meeting 2005*, September 2005.

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[4] E. Deelman et al., "Pegasus: Mapping Large Scale Workflows to Distributed Resources in Workflows in e-Science," *Springer*, 2006.

[5] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, vol. 15, no. 3, 2001.

[6] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: a Not-so-Foreign Language for Data Processing," in *Proc. 2008 ACM SIGMOD Conference on Management of Data*, 2008, pp. 1099–1110.

[7] F. Pentaris and Y. Ioannidis, "Autonomic Query Allocation Based on Microeconomics Principles," in *Proc. 23rd Int'l Conf. on Data Engineering (ICDE)*, 2007, pp. 266–275.

[8] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the Data: Parallel Analysis with Sawzall," *Sci. Program.*, vol. 13, no. 4, pp. 277–298, 2005.

[9] L. Vaguero and et al., "A Break in the Clouds: Towards a Cloud Definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, 1 2009.

[10] "Google Mashup Editor," code.google.com/gme/.

[11] "Microsoft Popfly," www.popfly.com.

[12] "Project DILIGENT," 2004, www.diligentproject.org.

[13] "Project D4Science," 2007, www.d4science.eu.

[14] "Project Health-e-Child," www.health-e-child.org.

[15] "The gCube System," www.gcube-system.org.

[16] "Yahoo! Pipes," pipes.yahoo.com/pipes/.

An Indexing Framework for Efficient Retrieval on the Cloud*

Sai Wu
National University of Singapore
wusai@comp.nus.edu.sg

Kun-Lung Wu
IBM T. J. Watson Research Center
klwu@us.ibm.com

Abstract

The emergence of the Cloud system has simplified the deployment of large-scale distributed systems for software vendors. The Cloud system provides a simple and unified interface between vendor and user, allowing vendors to focus more on the software itself rather than the underlying framework. Existing Cloud systems seek to improve performance by increasing parallelism. In this paper, we explore an alternative solution, proposing an indexing framework for the Cloud system based on the structured overlay. Our indexing framework reduces the amount of data transferred inside the Cloud and facilitates the deployment of database back-end applications.

1 Introduction

The emergence of the Cloud system has simplified the deployment of large-scale distributed systems for software vendors. The Cloud system provides a simple and unified interface between vendor and user, allowing vendors to focus more on the software itself rather than the underlying framework. Applications on the Cloud include Software as a Service system [1] and Multi-tenant databases [2]. The Cloud system dynamically allocates computational resources in response to customers' resource reservation requests and in accordance with customers' predesigned quality of service.

The Cloud system is changing the software industry, with far-reaching impact. According to an estimation from Merrill Lynch [3], by 2011, the Cloud computing market should reach $160 billion, including $95 billion in business and $65 billion in online advertising. Due to the commercial potential of the Cloud system, IT companies are increasing their investments in Cloud research. Existing Cloud infrastructures include Amazon's Elastic Computing Cloud (EC2) [4], IBM's Blue Cloud [5] and Google's MapReduce [6].

As a new computing infrastructure, the Cloud system requires further work for its functionalities to be enhanced. An area that draws most attention is data storage and retrieval. Current Cloud systems rely on underlying Distributed File Systems (DFS) to manage data. Examples include Google's GFS [8] and Hadoop's HDFS [9]. Given a query, the corresponding data are retrieved from the DFS and sent to a set of processing nodes for parallel scanning. Through parallel processing, the Cloud system can handle data intensive application efficiently. The challenges here lie in how to partition data among nodes and how to have nodes collaborate for a specific job. To simplify implementation, current proposals employ a simple query processing strategy, e.g.,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Partially supported by Singapore Ministry of Education (T2 grant) for UTab project

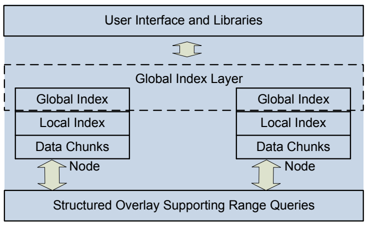


Figure 1: Indexing Framework of the Cloud

parallel scanning the whole data set. Given enough processing nodes, even the simple strategy can provide good performance. However, such an approach may only work in a dedicated system built for a specific purpose of a single organization. For example, Google employs its MapReduce [6] to compute the pagerank of web pages. In the system, nodes are dedicated to serving one organization. In contrast, in an open service Cloud system, such as Amazon's EC2, different clients deploy their own software products in the same Cloud system. Processing nodes are shared among the clients. Data management becomes more complicated. Therefore, instead of scanning, a more efficient data access service is required.

Following this direction, Aguilera et al.[7] proposed a fault-tolerant and scalable distributed B-tree for the Cloud system. In their approach, nodes are classified into clients and servers. The client lazily replicates all inner B^+-tree nodes, and the servers synchronously maintain a B^+-tree version table for validation. This scheme incurs high memory overhead for the client machine by replicating the inner nodes across the clients. Moreover, it is not scalable when the updates follow skewed distribution, invoking more splitting and merging on the inner nodes. In this paper, we examine the requirements for the Cloud systems and propose an indexing framework based on our earlier work outlined in [10]. Firstly, this indexing framework supports all existing index structures. Two commonly used indexes, hash index and B^+-tree index, are employed as examples to demonstrate the effectiveness of the framework. Secondly, processing nodes are organized in a structured P2P (Peer-to-Peer) network. A portion of the local index is selected from each node and published based on the overlay protocols. Consequently, we maintain a global index layer above the structured overlay. It effectively reduces the index maintenance cost as well as the network traffic among processing nodes, resulting in dramatic query performance improvement.

The rest of the paper is organized as follows: We present our indexing framework in the next section and discuss the details of our indexing approach in Section 3. In Section 4, we focus on the adaptive indexing approach. And some other implementation and research issues are introduced in section 5. Finally, we present our preliminary experimental results in Section 6 and conclude the paper in Section 7.

2 System Architecture

Figure 1 illustrates our proposed indexing framework for the Cloud system. There are three layers in our design. In the middle layer, thousands of processing nodes are maintained in the Cloud system to provide their computational resources to users. Users' data are partitioned into some data chunks and these chunks are disseminated to different nodes based on DFS protocols. Each node builds some local index for its data. Besides the local index, each node shares parts of its storage for maintaining the global index. The global index is a set of index entries, selected from the local index and disseminated in the cluster. The middle layer needs to implement the following interfaces:

Map(v)/Map(r)	Map a value or data range into a remote node
GetLI(v)/GetLI(r)	Given a value or data range, return the corresponding local index
GetGI(v)/GetGI(r)	Given a value or data range, return the corresponding global index
InsertGI(I)	Insert an index entry into the global index

All the methods except *GetLI* rely on the *Map* function. Given a value (hash based index) or a range (B^+-tree based index), *Map* defines how to locate a processing node responsible for the value or range. Its implementation depends on the lower layer's interface.

To provide an elegant interface for users, we apply the structured overlay to organize nodes and manage the global index. In the lower layer, processing nodes are loosely connected in a structured overlay. After a new node joins the Cloud, the node performs the join protocol of the overlay. Specifically, the node will accept a few other nodes as its routing neighbors and notify others about its joining. This process is similar to the construction of a P2P network. However, our system differs significantly from the P2P network. In the Cloud system, services are administrated by the service provider, and nodes are added into the system to provide computational resources. On joining the network, nodes must remain online unless the hardware fails. In contrast, in the P2P network, peer nodes are fully autonomous and unstable. A peer joins the P2P network for its own purpose (e.g., to download files or watch videos) and leaves the network on finishing its task. In our system, the P2P overlay is adopted only for routing purposes. The interfaces exposed for the upper layers are:

lookup(v)/lookup(r)	Given a value or a range, locate the responsible node
join	Join the overlay network
leave	Leave the overlay network

In principle, any structured overlays are applicable. However, to support B^+-tree based index, range search is required. Therefore, we adopt structured overlays that support range queries, such as CAN[11] and BATON[12].

In the upper layer, we provide a data access interface to the user's applications based on the global index. The user can select different data access methods for different queries. Scanning is suitable for the analysis of large data sets while index-based access is more preferred for online queries.

3 Indexing Framework

In this section, we shall discuss the implementation issues of the middle layer in the framework. Algorithm 1 shows the general idea of the indexing scheme. First, we apply an adaptive method to select some index values (the adaptive approach will be discussed in the next section). For a specific index value *v*, we retrieve its index entry through the *GetLI* method. The index entry is a value record in the hash based index or a tree node in the B^+-tree based index. Then, we apply the *Map* function to locate a processing node and forward the index entry to the node, where it will be added to the global index. Algorithm 2 shows the query processing algorithm via the global index. The query is forwarded to the nodes returned by the *Map* function, where the query is processed through the global index in parallel. As the algorithms show, the *Map* function plays an important role in the index construction and retrieval. In this section, we discuss how to define a proper *Map* function for different types of indexes.

Algorithm 1 EstablishGlobalIndex(node n)

```
1: ValueSet S=getIndexValue()
2: for ∀v ∈ S do
3:     I=GetLI(v)
4:     publish I to Map(v)
5: end for
```

3.1 Hash Based Indexing

The hash index is used to support exact key-match queries. Suppose we use the hash function h_l to build the local hash index. For an index value *v*, we can simply define the *Map* function as:

$$Map(v)=lookup(h_g(v))$$

Algorithm 2 SearchGlobalIndex(range r)

```
NodeSet N=Map(r)
for ∀n ∈ N do
    I=n.GetGI(r)
    process queries based on I
end for
```

where h_g is a global hash function for the Cloud system and *lookup* is the basic interface of the structured overlay. In the structured overlay, for routing purpose, each node is responsible for a key space. For the hash index, all nodes apply h_g to generate a key *k* for an index value. Given a key, *lookup* returns the node responsible for the key. Note that h_g does not need to be equivalent to the hash function h_l as each node may build their local hash index based on different hash functions.

3.2 B$^+$-tree Based Indexing

The B$^+$-tree based index is built for supporting range search. In an *m*-order B$^+$-tree, all the internal nodes, except the root node, may have *d* children, where $m \leq d \leq 2m$. The leaf nodes keep the pointers to the disk blocks of the stored keys. To define the *Map* function for the B$^+$-tree index, a range is generated for each tree node. Basically, B$^+$-tree nodes can inherit a range from their parents. In Figure 2, node d is node a's third child. So its range is from the second key to the upper bound of a, namely (35,45). The range of *a* is from the lower bound of the domain to the first key of its parent. Thus, *a*'s range is (0,45). Specifically, the range of the root node is set to be the domain range.

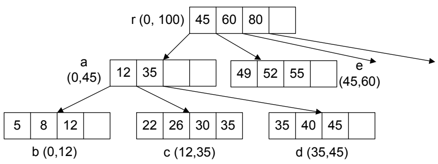


Figure 2: Node Range in B$^+$-tree

After generating the range for a B$^+$-tree node *n*, we define the *Map* function as:

Map(n)=lookup(range(n))

To support the above mapping relation, the underlying overlay must provide the *lookup* interface for a specific range. In this case, only the structured overlays that support range search are applicable, such as BATON [12], CAN [11] and P-Ring [13].

3.3 Multi-dimensional Indexing

A multi-dimensional index, such as the R-tree [14], is useful for spatial and multi-dimensional applications. In the R-tree, each node is associated with a Minimal Boundary Rectangle (MBR), which is similar to the range defined for the B$^+$-tree node. Given an R-tree node, we need to define a *Map* function to locate the processing node. Depending on the characteristics of the underlying overlays, we have two solutions:

If the underlying overlay, such as CAN [11], supports multi-dimensional routing, we can directly use its *lookup* interface. For an R-tree node n, the *Map* function is defined as:

Map(n)=lookup(getMBR(n))

However, most structured overlays have not been designed for supporting multi-dimensional data indexing. In this case, the alternative solution is to map the multi-dimensional rectangle into a set of single dimensional ranges. The space filling curve [15] is commonly used for this task. Given a rectangle R, we can define a function f based on the space filling curve, which maps R to a range set S. Finally, the *Map* function returns the corresponding node set:

$$Map(n)=\{lookup(r)|\forall r \in S\}$$

4 Index Tuning

The local index size is proportional to the data size. Therefore, we cannot publish all the local indexes into the global index. In this section, we discuss the index tuning problem in the framework.

Algorithm 3 IndexTuning(node n)

```
IndexSet I=n.getAllIndexEntry()
for ∀e ∈ I do
    if needSplit(e) then
        IndexSet I'=getLowerLevelIndexEntry(e)
        remove e and insert I' into global index
    else
        if needMerge(e) then
            IndexEntry e'=getUpperLevelIndexEntry(e)
            remove e and its siblings; insert e' into global index
        end if
    end if
end for
```

Algorithm 3 shows the general strategy of index tuning. If an index entry needs to be split due to the high benefit for query processing, we replace the index entry with its lower level index entries. In contrast, if it needs to be merged with its siblings, we remove all the corresponding index entries and insert their upper layer entry. In this way, we dynamically expand and collapse the local index in the global index. In the above process, we manage the local index in a hierarchical manner. Existing index structures can be easily extended to support such operations. Again, we use hash index and B$^+$-tree index as the examples in our discussion.

4.1 Multi-level Hash Indexing

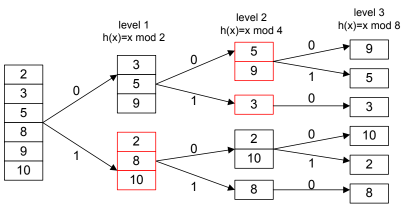


Figure 3: Hierarchical Hash Functions

Linear hashing and extendible hashing can be considered as multi-level hash functions. As shown in Figure 3, the hash function at level *i* is defined as *h(x)=x mod* 2^i. Given two data items v_1 and v_2, if $h_i(v_1) = h_i(v_2)$, v_1 and v_2 are mapped to the same bucket in level *i*. As a matter of fact, the index data are only stored in the buckets of the last level (e.g., level 3 in Figure 3). The other level buckets store a Bloom Filter [16] to verify membership and are maintained virtually. We generate an ID for each bucket based on its ancestors' hash values. For example, the bucket $B_i = \{3, 9\}$ in level 2 has an ID "00" and the bucket $B_j = \{8\}$ in level 3 has an ID "110". Instead of using the hash value as the key to publish the data, we use the bucket ID as the key. Initially, only level 1 buckets (e.g., bucket "0" and "1") are inserted into the global index. If bucket 0 has a high query load, it will be split into two buckets in level 2. Then, the query load is shared between the two buckets. The index lookup is performed in a similar way. We generate a search key based on the hash function. For example, to perform search for 9 and 6, we generate keys “000” and “100”, respectively. Query for “000” will be sent to the bucket “00”, whose id is the prefix of the query.

4.2 Dynamic Expansion of the B$^+$-tree based Indexes

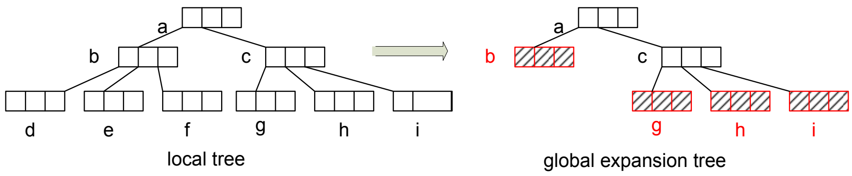


Figure 4: Adaptive Expansion of B$^+$-tree

In the index tuning process, the B$^+$-tree based global index can be considered a result of the dynamic expansion of the local B$^+$-trees. Figure 4 illustrates the idea. Due to network cost and storage cost, we cannot publish all the leaf nodes into the global index. Therefore, it is more feasible and efficient to select and publish some tree nodes based on the cost model. Based on Algorithm 3, the tuning process is similar to tree expansion or collapse. When a new processing node joins the cluster, it inserts the root node of its local B$^+$-tree into the global index. Then, it adjusts its index by expanding the tree dynamically. Figure 4 shows a snapshot of an expanding tree.

4.3 Cost Modeling

In our indexing framework, a cost model is essential to evaluate the cost and benefit of maintaining the global index. In As different system configurations will lead to different cost models, we describe a general approach to estimate the cost. Basically, maintenance costs can be classified into two types, query processing cost and index maintenance cost. Algorithm 2 indicates that query processing cost includes the routing cost incurred by *Map* and the index lookup cost incurred by *GetGI*. Based on the protocol of structured overlays, the cost of *Map* is $O(logN)$ network I/O, where N is the number of nodes in the Cloud. The cost of *GetGI* is the local I/O cost of processing the query via the global index, and it depends on the structure of current global index. For example, in an L-level B$^+$-tree index, if one h-level tree node n_i is inserted into the global index, query processing via n_i requires additional $L - h$ I/O cost. Thus, the cost of *GetGI* must be estimated on the fly. Once the local index is modified, we need to update the corresponding global index. A typical update operation triggers $O(logN)$ network I/Os and some local I/Os. The total index maintenance cost is a function of the update pattern. We employ the random walk model and the bayesian network model to predict update activities in the B$^+$-tree index and the multi-level hash index, respectively. Finally, the cost of a specific index entry is computed as the sum of its query cost and maintenance cost. And to limit the storage cost, we set a threshold for the size of global index. Then, the optimal indexing scheme is transformed into a knapsack problem. And a greedy algorithm can be used to solve the problem.

5 Other Implementation Issues

5.1 Concurrent Access

In an open service Cloud system, registered users are allowed to deploy their own softwares. If some users' instances access the global index concurrently, we need to guarantee the correctness of their behaviors. Suppose an index entry receives an update request and read request simultaneous from different instances. We need to generate a correct serialized order for the operations. A possible solution is to group the relative operations in a transaction and apply the distributed 2-phase locking protocol. However, 2-phase locking protocol reduces the performance significantly. If consistency is not the major concern, more efficient solutions may be possible [17].

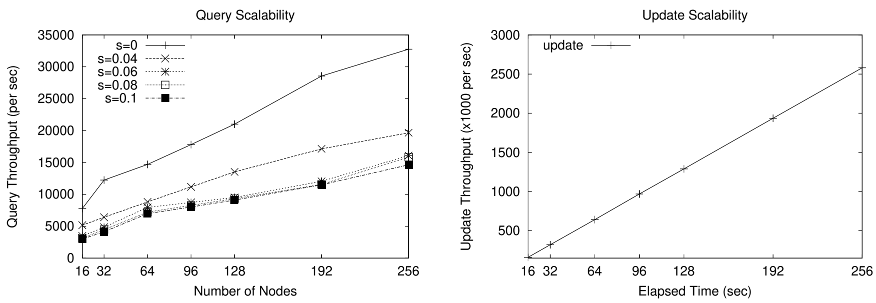
5.2 Routing Performance

As discussed in the cost model, *Map* incurs $O(logN)$ network I/O, where N is the number of nodes in the Cloud. Although nodes in the Cloud are connected via a high bandwidth LAN, the network cost is still dominating the index lookup cost. Some systems [18] apply the routing buffer to reduce the network cost. Generally, after a success lookup operation, the node keeps the destination's metadata in its local routing buffer. In the future processing, if a new lookup request hits the buffer, we can retrieve the corresponding data within 1 network I/O. However, the application of routing buffer incurs new research problems such as how to keep the routing buffer up to date and how to customize the routing algorithm.

5.3 Failure Recovery

In the Cloud system, as the processing nodes are low-cost workstations, there may be node failures at any time. In this case, a master node is used to monitor the status of nodes. And each node will record its running status into a log file occasionally. If a node fails, it will be rebooted by the master node and automatically resume its status from the log file. To keep the high availability of the global index, we write the global index into the log file as well. Moreover, we exploit the replication protocol of the overlay network to create multiple copies of the global index. Therefore, a single node's failure will not affect the availability of the global index. One of the replicas is considered as the master copy, while the other are slave copies. The updates are sent to the master copy and then broadcasted to the slave copies. Once a master copy fails, one of the slave copies is promoted to be the master one. And after a node recovers its global index via the log file, it will become a slave copy and ask the master one for the missing updates.

6 A Performance Evaluation



(a) Query Throughput (b) Update Throughput

Figure 5: Experiment Result

To illustrate the effectiveness of the framework, we have implemented our indexing framework on BATON [12] for the Cloud system (see [10] for more details). In our Cloud system, each node builds a local B^+-tree index for its data chunks. The global index is composed by a portion of local B^+-tree indexes. We deploy our system on Amazon's EC2 [4] platform. In our system, each node hosts 500k data in its local database. A simulator is employed to issue queries. From the start of the experiment, the node will continuously obtain a new query from the simulator after it finishes its current one. The major metrics in the experiment are query throughput and update throughput. To test the scalability of our approach, Cloud systems with different numbers of processing nodes are created. In Figure 5(a), we generate different query sets by varying the selectivity of the search. When $s = 0$, the query is exact search query. When $s = 0.01$, one percent of the data space is searched in the query. Query throughput increases almost linearly as the number of processing nodes increases. Figure 5(b) shows the update throughput. We generate the insertion request for each local B^+-tree uniformly. In our system, the updates can be processed by different nodes in parallel.

7 Conclusions

In this paper, we study and present a general indexing framework for the Cloud system. In the indexing framework, processing nodes are organized in a structured overlay network, and each processing node builds its local index to speed up data access. A global index is built by selecting and publishing a portion of the local index in the overlay network. The global index is distributed over the network, and each node is responsible for maintaining a subset of the global index. Due to storage cost and other maintenance issues, an adaptive indexing approach is used to tune the global index based on the cost model. Two experiments on a real Cloud environment, Amazon's EC2, illustrate the effectiveness and potential of the framework.

References

[1] Steve Fisher. Service Computing: The AppExchange Platform. *SCC*, 2006.

[2] M. Hui and D. W. Jiang and G. L. Li and Y. Zhou. Supporting Database Applications as a Service. *ICDE*, 2009.

[3] Merrill Lynch. The Cloud Wars: $100+ billion at stake. 2008.

[4] Merrill Lynch. Amazon Elastic Compute Cloud (Amazon EC2) http://aws.amazon.com/ec2/.

[5] IBM. IBM Introduces Ready-to-Use Cloud Computing, http://www−03.ibm.com/press/us/en/pressrelease/22613.wss.

[6] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 2008.

[7] Marcos Aguilera and Wojciech Golab and Mehul Shah. A Practical Scalable Distributed B-Tree. *VLDB*, 2008.

[8] Sanjay Ghemawat and Howard Gobioff and Shun-Tak Leung. The Google file system. *SOSP*, 2003.

[9] http://hadoop.apache.org

[10] Sai Wu and Dawei Jiang and Beng Chin Ooi and Kun-Lung Wu. CG-index: A Scalable Indexing Scheme for Cloud Data Management Systems. *Technique Report (http://www.comp.nus.edu.sg/ wusai/report_09_01.pdf)*, 2009.

[11] Sylvia Ratnasamy and Paul Francis and Mark Handley and Richard Karp and Scott Schenker. A scalable content-addressable network. *SIGCOMM*, 2001.

[12] H. V. Jagadish and Beng Chin Ooi and Quang Hieu Vu. BATON: A Balanced Tree Structure for Peer-to-Peer Networks. *VLDB*, 2005.

[13] Adina Crainiceanu and Prakash Linga and Ashwin Machanavajjhala and Johannes Gehrke and Jayavel Shanmugasundaram. P-ring: an efficient and robust P2P range index structure. *SIGMOD*, 2007.

[14] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD*, 1984.

[15] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the Hilbert space-filling curve. *SIGMOD Record*, 30(1), 2001.

[16] Andrei Broder and Michael Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 2002.

[17] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2), 51-59, 2002.

[18] Giuseppe DeCandia and Deniz Hastorun and Madan Jampani and Gunavardhan Kakulapati and Avinash Lakshman and Alex Pilchin and Swaminathan Sivasubramanian and Peter Vosshall and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS*, 2007.

25th IEEE International Conference on Data Engineering (ICDE 2009)
29 March – 4 April, 2009 Shanghai, China

Data Engineering refers to the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The **25th International Conference on Data Engineering** provides a premier forum for sharing and exchanging research and engineering results to problems encountered in today's information society. The conference programme will include research papers on all topics related to data engineering, including but not limited to:

Approximation and uncertainty in databases	**Social information management, annotation and data curation**
Probabilistic databases	**Query processing and query optimization**
Data integration	**Database tuning, and autonomic databases**
Metadata management and semantic interoperability	**Scientific, biomedical and other advanced applications**
Data mining and knowledge discovery	**Spatial, temporal and multimedia databases**
Data privacy and security	**Transaction and workflow management**
Data streams and sensor networks	**Ubiquitous, mobile, distributed, and peer-to-peer databases**
Data warehousing, OLAP and data grids	**Web data management**
Database user interfaces and information visualization	**XML data management**
Personalized databases	**Database architectures**

Accepted contributions at ICDE 2009 will make efforts (1) to expose practitioners to the most recent research results, tools, and practices that can contribute to their everyday practical problems and to provide them with an early opportunity to evaluate them; (2) to raise awareness in the research community of the difficult data & information engineering problems that arise in practice; (3) to promote the exchange of data & information engineering technologies and experiences among researchers and practitioners; and (4) to identify new issues and directions for future research and development in data & information engineering.

AWARDS

An award will be given to the best paper submitted to the conference. A separate award will be given to the best student paper. Papers eligible for this award must have a (graduate or undergraduate) student listed as the first and contact author, and the majority of the authors must be students.

INDUSTRIAL PROGRAM

ICDE 2009 will include an industrial track covering innovative commercial implementations or applications of database or information management technology, and experience in applying recent research advances to practical situations. Papers will describe innovative implementations, new approaches to fundamental challenges (such as very large scale or semantic complexity), novel features in information management products, or major technical improvements to the state-of-the-practice.

PANELS

Conference panels will address new, exciting, and controversial issues, being provocative, informative, and entertaining.

DEMONSTRATIONS

Presented research prototype demonstrations will focus on developments in the area of data and knowledge engineering, showing new technological advances in applying database systems or innovative data management/processing techniques.

TUTORIALS

ICDE 2009 will host tutorials, relevant to the conference topics. Tutorials can be single-session (1.5 hour) or for double-session (3 hour).

WORKSHOPS

The following workshops will be hosted by ICDE 2009:

- **DBRank: Third International Workshop on Ranking in Databases**
- **First IEEE Workshop on Information & Software as Services (WISS'09)**
- **Fourth International Workshop on Self-Managing Database Systems (SMDB 2009)**
- **Management and Mining of UNcertain Data (MOUND)**
- **Modeling, Managing, and Mining of Evolving Social Networks (M3SN)**
- **Second International Workshop on Data and Services Managementin Mobile Environments (DS2ME 2009)**

For more information, visit http://i.cs.hku.hk/icde2009/

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398