Energy Efficient Data Intensive Distributed Computing

Weiwei Xiong
University of California San Diego
La Jolla, CA 92093

Aman Kansal
Microsoft Research
Redmond, WA 98052

Abstract

Many practically important problems involve processing very large data sets, such as for web scale data mining and indexing. An efficient method to manage such problems is to use data intensive distributed programming paradigms such as MapReduce and Dryad, that allow programmers to easily parallelize the processing of large data sets where parallelism arises naturally by operating on different parts of the data. Such data intensive computing infrastructures are now deployed at scales where the resource costs, especially the energy costs of operating these infrastructures, have become a significant concern. Many opportunities exist for optimizing the energy costs for data intensive computing and this paper addresses one of them. We dynamically right size the resource allocations to the parallelized tasks such that the effective hardware configuration matches the requirements of each task. This allows our system to amortize the idle power usage of the servers across a larger amount of workload, increasing energy efficiency as well as throughput. This paper describes why such dynamic resource allocation is useful and presents the key techniques used in our solution.

1 Introduction

Data intensive distributed computing platforms such as MapReduce [4], Dryad [7], and Hadoop [5], offer an effective and convenient approach to solve many problems involving very large data sets, such as those in web-scale data mining, text data indexing, trace data analysis for networks and large systems, machine learning, clustering, machine translation, and graph processing. Their usefulness has lead to widespread adoption of such data intensive computing systems for data analysis in many large enterprises, and server clusters consisting of tens of thousands of servers now host these platforms. At these scales, the cost of resources consumed by the data parallel computing system becomes very significant. Both the operational costs such as energy consumption, and the capital expense of procuring the servers and supporting infrastructure are thus crucial to optimize.

Several opportunities exist in data intensive distributed computing systems to maximize the amount of work performed by a given set of resources and to reduce the energy consumed for performing a given amount of work. A first opportunity is to schedule the processing jobs in a manner that is best suited to the data placement and network requirements of each job, resulting in improved utilization of the infrastructure [8]. Another possibility is to reduce the amount of energy spent on storage. While redundant copies of data are required for reliability, machines hosting some of the redundant copies can be shut down if the throughput requirement and data consistency constraints permit [9]. A third opportunity arises in adapting the resource configuration to the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

resource needs of the currently processing jobs. In this paper we consider this last opportunity in detail and show how energy use can be reduced and throughput increased for a given set of tasks and cluster size.

If the server cluster was to always run a single, pre-specified, data processing job, then it would be straightforward, though possibly tedious, to configure the server hardware to match the requirements of that processing task efficiently. However, in practice a single cluster is set up for multiple data analysis jobs. In fact, sharing the cluster yields significant advantages in efficient resource utilization since one job is unlikely to always have sufficient workload to justify a dedicated cluster. The complication that arises due to sharing is that the server configuration is not optimized to meet the resource requirements of any single job. At run time, depending on the characteristics of the job, certain resources such as the number of disks spinning or active servers, would be left unused, wasting valuable power and also the capital expense of the infrastructure itself. Since it is impractical to change the hardware dynamically, we explore software mechanism to optimize resource utilization and improve overall throughput, using Dryad as the underlying data intensive computing platform.

Specifically, we introduce a mechanism, referred to here as Energy Efficient Dryad (e-Dryad, for short), for fine grained allocation of cluster resources to incoming jobs that *right-sizes* the resource configuration to a job's requirements. Current job schedulers for data intensive distributed computing platforms allocate a set of servers (sometimes referred to as nodes) to each job. The number of servers allocated depends on the job's requirements, its priority, and the number of currently unallocated servers. However, the hardware configuration of the servers may not be matched to the processing demands of the job. Our proposed mechanism divides the server resources among jobs to match their hardware resource usage characteristics, leading to a more efficient and better balanced resource configuration for each job.

While e-Dryad improves the overall throughput compared to existing schedulers, the amount of instantaneous resources allocated to a job may in fact be smaller in our proposed approach. This does cause the latency of processing to to be higher in certain situations. However, the increased throughput implies that the jobs spend less time waiting for their turn in the queue and hence the overall completion time can in fact still be lower depending on the position in the queue.

We describe the design of e-Dryad, along with the challenges addressed and mechanisms used, in Section 2. The proposed system is evaluated with real workloads on an experimental server cluster, and the evaluation results are presented in Section 3. Section 4 discusses related problems and possible extensions. Section 5 summarizes the prior works in the area and Section 6 concludes.

2 Resource Allocation System Design

The problem of inefficient resource utilization arises because the hardware is not configured to match the resource requirement of every job. We explain this problem in more detail below and then present a solution to address it.

2.1 Problem Description

We describe the problem in the context of Dryad, since that is the platform used in our prototype, but the problem conceptually exists in most data intensive computing platforms. A common operational aspect of such platforms includes a scheduler that accepts computing jobs and assigns them to servers. A *job* here refers to a data processing program that includes at least one processing phase that can be massively parallelized. It may be followed by additional processing stages that further process or integrate the results from the parallelized phase or phases. In Dryad, each job is comprised of a root task that manages the actual processing tasks, referred to as worker tasks. The worker tasks accesses data stored on the server cluster's constituent computers themselves. The *scheduler* queues incoming jobs and as servers become available after finishing previously running jobs, they are assigned to new jobs from the queue. A good scheduler will assign servers to a job's worker tasks such

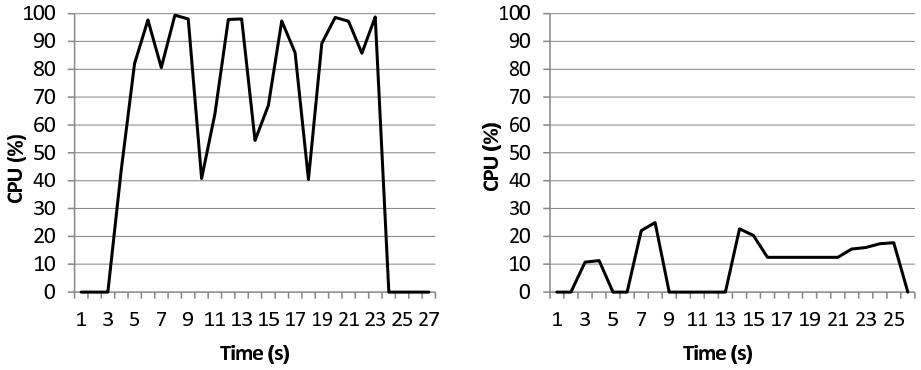
that the data they access from storage is on the assigned server itself or relatively close by within the network topology [8].

The parallelized phase is the one that requires the most resources and we primarily focus on the worker tasks comprising this phase. The worker tasks within a job are largely homogeneous in terms of computational resource requirement but the worker tasks from different jobs can be very different. Consider the following two examples of real world applications that are well suited to parallelized data intensive computation:

Word Statistics. This application computes the number of times different words occur in a very large corpus of text, such as a large number of web pages, stored across multiple machines [11]. Such tasks are commonly required for web scale indexing.

Terasort. The Terasort application sorts a very large number (billions) of records using case insensitive string comparisons on a 10-byte key [10]. This is a benchmark used for parallel computing.

Figure 1 shows the resource utilization for CPU, averaged across multiple servers, for a job from each of the above applications.



(a) Word Statistics.

(b) Terasort.

Figure 1: Resource usage of two Dryad jobs, averaged across multiple machines allocated during the job's execution.

It is clear from the figure that the Word Statistics job has a significantly higher CPU utilization. A similar figure may be plotted for IO bandwidth usage and that would show that the Terasort job has a higher IO usage and is not bottlenecked on CPU. Each job thus leaves some resource wasted. This is a problem due to two reasons. Firstly, the cost of the infrastructure directly depends on the number of servers required, and hence if the servers are left unused, that results in wasted resources. In the above example, the unused CPU could potentially have been used by other jobs waiting in the queue. Secondly, the above usage causes energy wastage. A large fraction of the power usage of a server, often as high as 60%, is spent to simply power on a server, and is referred to as idle power. This power is spent even if the CPU or other resources are not used by any job. Clearly, if the CPU resource left unused by Terasort were to be used by another job, the idle energy use will not increase and only a small amount of additional energy would be required. When resource utilization is high, the idle energy is amortized over greater amount of work done, resulting in more efficient operation.

Our goal in building e-Dryad is to address precisely the above two problems by improving resource utilization leading to better amortization of idle energy, and higher throughput from the given number of servers.

2.2 Resource Allocation Methodology

The basic idea for resource allocation in e-Dryad is to place worker tasks from multiple jobs on each machine, in a manner that improves the resource utilization. This implies that the worker tasks placed together must be carefully chosen. If two worker tasks are both bottlenecked on the resource, such as CPU, then placing them together would simply increase the run time of each task without improving resource utilization significantly. Instead, if two worker tasks with complimentary resource utilizations are placed on the same server, then the overall resource utilization would improve. Suppose one task is bottlenecked on CPU, while another is bottlenecked on the storage bandwidth, then placed together, they could utilize both resources maximally, leading to better amortization of the idle power. An important consideration here is that each task needs both storage and CPU resource. The CPU intensive task is also ultimately a data intensive program and would need some storage bandwidth and the storage intensive task obviously needs some CPU capacity to perform its work. This implies that the resource allocation mechanism should ensure that each task gets resources in its desired proportion. This may be viewed as re-balancing the hardware configuration to match the job requirements.

The e-Dryad resource allocation mechanism, based on the above basic idea, is shown in Figure 2, and described below.

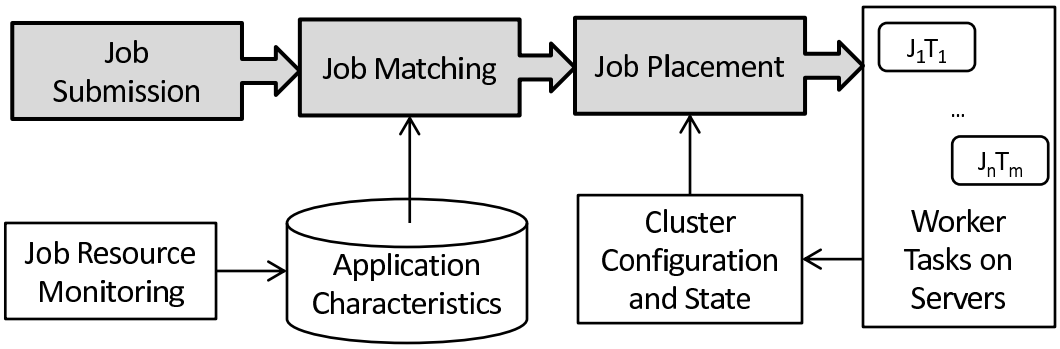


Figure 2: e-Dryad resource allocation block diagram

All incoming job requests are accepted through the *job submission* system as usual. In Dryad, jobs are often submitted through a web interface provided for the Dryad cluster, by uploading a job specification.

2.2.1 Job Matching

The first crucial step for e-Dryad is matching job resource usage characteristics to determine complementary jobs that are well suited to be placed together. In data intensive computing clusters, while new jobs are submitted everyday to process new data sets, the nature of the jobs themselves does not change as frequently. Rather, a slowly evolving set of applications is hosted by the cluster. We assume that repeat instances of the same application each have a unique job ID but are labeled with an application ID that allows identifying them as different instances of the same application. The e-Dryad resource allocation system maintains a set of *application characteristics*, shown as an input to the job matching block in the figure.

The application characteristics are acquired by monitoring the resource usage of multiple jobs belonging to the same application. The characteristics may be initialized based on the first instance, and updated over time as additional jobs from that application are observed. Monitoring of job resource usage is performed by the *job resource monitoring* block shown in the figure. This block itself is a distributed monitoring system that uses the Dryad infrastructure to correctly track the distributed resource usage of each job across multiple machines. The implementation of this system in our prototype is discussed in Section 2.3.

Based on the application characteristics, the job matching block identifies the resource for each application that shows the heaviest usage. This resource is likely the bottleneck for this application and hence the other resources on a machine running this application jobs would have spare capacity. Note that the determination of the bottleneck is only an estimate since the heavy resource usage may be an effect of data placement and network topology that may change based on job placement. The estimate improves over time as a well designed scheduler attempts to place each job's tasks efficiently in terms of data placement and the application characteristics are then dominated by the true bottlenecks. Matched jobs are ones that have a different bottleneck resource, and can hence be placed together to reduce the idle power consumption per unit work.

2.2.2 Job Placement

Placement of jobs determines the actual allocation of resources to the worker tasks of the jobs. Once the job matching block has determined which jobs are suitable to be placed together to optimize resource utilization, the job placement block assigns the worker tasks to the servers in the cluster based on the current state of the cluster in terms of server availability and other scheduling considerations. The cluster configuration and state information is directly measured from the cluster management interfaces.

Multiple different job placement heuristics may be employed based on the nature of jobs and desired performance and energy optimization objectives. We assume that the scheduling mechanism is the same as followed in Dryad and abstract it out, focusing only on the placement. A key resource allocation decision for placement after matching jobs are identified, is to determine how the resources of the server should be divided among the tasks. In our prototype we focus on the two resources that affect throughput the most for data intensive computation in a well scheduled system: processor time and storage bandwidth. The following two heuristics are employed for resource allocation:

Equal Sharing. In the equal sharing approach, two tasks from different jobs that are placed on a common server, are each assigned an equal share of the CPU time. One of these tasks is CPU intensive and leaves significant storage bandwidth unused. The other task being storage intensive, uses up a large fraction of the storage bandwidth even with half its usual CPU allocation since it is not bottlenecked on the CPU. The overall amount of work performed increases leading to improved amortization of idle energy.

Proportional Sharing. In the proportional sharing approach, rather than dividing the CPU resource equally, it is divided in a ratio proportional to the average CPU utilization known for the two jobs. While the CPU usage can vary across tasks and over the run time of a job, the division in proportion to the average usage captures the resource requirements better than equal sharing. In this case, resource utilization is further improved since the tasks bottlenecked on CPU can use up a greater portion of the CPU while the other task can likely get a sufficient amount of storage bandwidth even with a smaller share of the CPU. The proportional sharing does not guarantee that resource will be maximally utilized since it is unlikely that the resource requirements of different tasks are matched exactly in different resource dimensions to use each resource to a 100%.

The relative performance of the two approaches is studied experimentally in Section 3.

2.3 Implementation

The implementation of e-Dryad is based on a Dryad cluster installed on Windows Server 2008 systems. The operation of e-Dryad requires detailed resource usage tracing for job characterization as well as experimental verification. The job resource monitoring component, referred to in Figure 2 earlier, is implemented as follows. Tracking the resource usage of a single job requires understanding the placement of the tasks comprising the job on various servers, and then tracking the resource utilization of each task on each of those servers. The

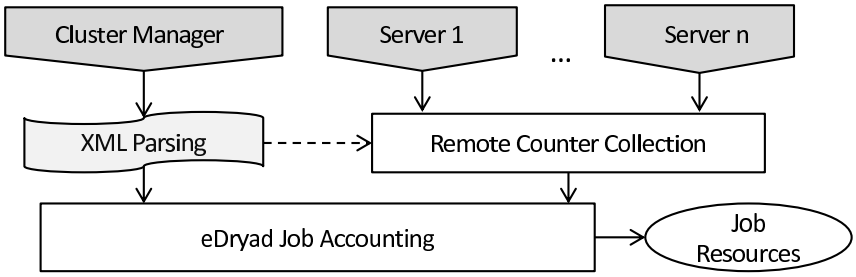


Figure 3: e-Dryad job resource monitoring.

monitoring system in e-Dryad is implemented as shown in Figure 3. The cluster manager in Dryad produces an XML formatted description of job layout across the servers. e-Dryad fetches that description to infer the servers allocated to each job at each time step. Resource usage at each server is monitored for the Dryad processes using Windows kernel provided performance counters. Windows system management enables these performance counters to be remotely accessed and e-Dryad accesses the counters for each of the processes from multiple remote servers. Based on the job to server mapping, it accounts for the resource usage of each job.

The system mechanism used to assign the right amount of resource is based on server virtualization. We create virtual machines on each server and divide the physical resources among virtual machines in a proportion that suits the job placement across servers. For instance, going back to the example shown in Figure 1, suppose quad core servers were used, one virtual machine could be assigned three processor cores and allocated to Word Statistics job, while the other one could be given one processor core and allocated to the Terasort job. Clearly, the bandwidth on the machine would also get split among the two VMs. Terasort may use a greater share of the bandwidth based on its needs. The virtualization based mechanism integrates seamlessly with the Dryad scheduler since from the perspective of the scheduler, each virtual machine is a different server to be assigned to a job. However, virtualization is only one of the options to allocate resources in the correct proportion. Equivalent functionality could also be achieved by simultaneously assigning the same server to multiple jobs and creating separate processes to host each job. The Dryad scheduler may be modified to incorporate e-Dryad based job characteristics in determining the job allocations across servers. The operating system on each allocated server could then be informed to allocate resources in the correct proportion to the different processes running the worker tasks from different jobs on each server.

3 Evaluation

We evaluate e-Dryad on a 32 node cluster, with Intel Xeon 2.5GHz processors. We use data intensive applications found in Dryad literature [11], including the two applications mentioned in Section 2.1, that are generally representative of data indexing tasks. The jobs are specified using DryadLINQ [10].

The energy use and savings for the equal and proportional sharing policies is shown in Table 1, and also compared to that of placement without e-Dryad. The improved job placement leads to significant energy savings, that for large scale data intensive computing clusters result in significant cost savings.

While the energy savings are definitely a desirable feature of e-Dryad and come along with increased overall throughput, they are also associated with a potential increase in latency, in terms of job execution time measured from the instant that a job is allocated on the cluster. The overall latency, including the queuing delay and processing delay may not be increased for most jobs. However, high priority jobs, that were not subject to significant queuing delays in the default design, could suffer increased latency. This is illustrated in Figure 4

Placement	Energy Use (J)	Energy Saving (%)
Dryad-default	35295	–
Equal Sharing	29303	17.01
Proportional Sharing	28282	20.40

Table 1: Energy Savings achieved by e-Dryad in experimental tests.

taking two jobs, one from the word statistics application and the other from the Terasort application. The figure

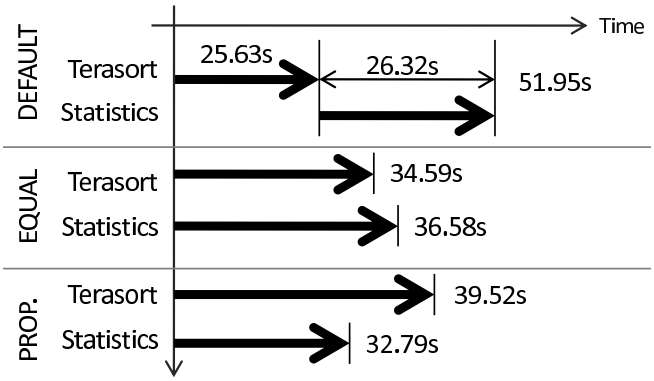


Figure 4: Illustration of effect on job latency with e-Dryad resource allocation. The "Prop." label refers to the proportional sharing approach.

shows the execution of these two jobs along the time line. The top segment shows how execution proceeds in the default system and the lower two segments show how execution proceeds with e-Dryad policies. Each job consumes a shorter processing time in the default system but the total execution time on the system is longer. For the first job the latency is smaller but the second job suffers a longer latency due to longer queuing delay. The simultaneous execution in the other two approaches allows the two jobs to exit the system sooner, leading to a lower total energy cost and higher throughput. The latency of the Terasort job has however increased by 34.9% in the equal sharing case and 54.2% in the proportional sharing case, due to reduced instantaneous resources granted.

The resource utilization also improves in the system. Taking the above run as an example, the CPU utilization is plotted in Figure 5. The utilizations shown are averaged across the 8 worker tasks comprising each job. As seen, the utilization is higher during the run with e-Dryad, and the jobs finish sooner (at 39s instead of 52s).

4 Discussion

The e-Dryad approach for resource allocation in data intensive computing highlights an important opportunity for improving energy costs and infrastructure expense. However, there are several additional features and design extensions that may be considered to the initial design presented above. We discuss some of these extensions below.

Temporal Profiling: In e-Dryad, we considered job level characteristics. A job has multiple stages of processing. The worker tasks perform different types of computation in each stage and more accurate resource utilization characteristics would emerge if the worker tasks of different stages were characterized individually. Further, even among the homogeneous parallel tasks of a single stage, resource usage may differ because of

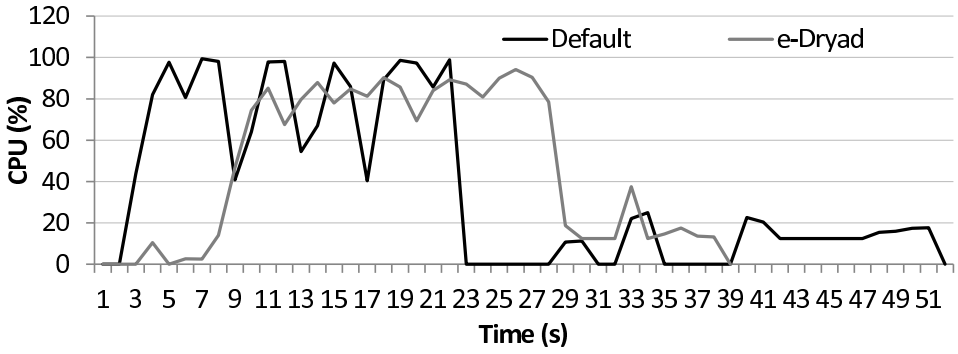


Figure 5: Resource utilization with and without e-Dryad.

placement. Some tasks may be accessing only local data, some could be fetching their data from remote machines, and some could be using a mix of local and remote data. Differences in network data rates available to different tasks will lead to differences in each task's server resource usage and considering such effects can help further optimize the e-Dryad resource allocation. However, in pactice, acquiring such detailed characteristics, some of which depend on run time placement, is often difficult.

Latency Constraints: Joint placement of jobs for improved resource efficiency sometimes requires reducing the instantaneous resource allocation for a job, and we showed in evaluations that the processing latency of a job, measured from the time it is allocated on the cluster, may be increased. This may happen for instance, when the storage bandwidth on the machines containing a job's data is shared with another job to improve the CPU utilization. In certain scenarios, we may wish the first job to have the entire storage bandwidth to minimize its latency. To address such scenarios, e-Dryad resource allocation may be modified to incorporate priorities and the high priority jobs may be allocated greater resources, at the cost of overall cluster efficiency.

Throughput Trade-off: We saw for the experimental system configurations and resource allocation policies that energy efficiency also improved throughput, though sometimes at the cost of higher latency. There are resource allocation policies when energy efficiency may be improved further by reducing the throughput. Given that the idle power of keeping a server powered on is high, we optimized energy efficiency by maximally utilizing available disk throughput. However, consider a policy where some of the servers are turned off, to ensure that all data is available but multiple redundant copies may not be on line. This will allow for reducing the idle energy costs but also reduce the available IO bandwidth. If all the jobs are processor intensive (IO bandwidth was not the bottleneck) and the reduced IO bandwidth does not cause the processors on powered on servers to stall on data access, then this reduced server configuration will be better aligned to job resource requirements and hence more energy efficient. In such a policy, that exploits additional resource management knobs such as server shutdown, both throughput and latency may be traded-off for improved energy efficiency.

Dynamic Allocations: The job allocation policy discussed in our prototype assumes that the resource allocation is static for each job. This is reasonable when the job queues are typically never empty and jobs are served in the order received. However, in cases where high prioity jobs require pre-emption of currently allocated jobs, or the arrival of a new job opens up opportunities for more efficient resource allocations than the current allocation, it becomes worthwile to consider dynamic resource allocations. If the tasks corresponding to a running job are stopped, the work performed until that time would be wasted. Dynamic allocation would thus require the capability to suspend a running task by saving the intermediate results and task state.

Multiple Resources: The heuristics presented for job matching and placement considered only two resources and shared a server among at most two worker tasks from matched jobs. The design may be extended to additional resources and may place more than two tasks on the shared server to further improve resource usage in even more dimensions leading to lower energy use per unit throughput.

5 Related Work

The problem of energy efficiency in data intensive computing platforms is a relatively new area of research. Another work directly addressing this problem, though through a different approach is presented in [9]. The placement of data among the servers is optimized such that certain servers may be placed in low power states when feasible.

Simultaneous placement of multiple workloads on the same servers is also considered in several works focused on virtual machine consolidation, such as [1,6]. However, the key problem addressed in virtual machine consolidation is efficient packing along pre-specified resource dimensions such as CPU time and memory space. The bottleneck resources of data intensive computing tasks, such as storage bandwidth, are not considered. Also, the resource requirements are assumed specified, while in our work we employ distributed monitoring to characterize the relevant resource characteristics of applications. The e-Dryad method also trades off latency for energy efficiency while virtual machine consolidation typically focuses on optimizing performance.

Another set of related works is found in optimizing the performance of data intensive computing platforms such as by improving the schedulers for Dryad or MapReduce [3, 8, 12] for addressing various challenges such as efficient handling of failed tasks, adaptation to data and network topology, and better utilization of cache hierarchy. The optimization of the jobs themselves through static analysis was proposed in [2]. Our work is complementary to the above works and could potentially be added on as a resource re-configuration layer with many of their scheduling policies. e-Dryad improves resource efficiency by matching the resource allocation to the resource configurations best suited to the computational tasks.

6 Conclusions

We described an important opportunity for improving the energy efficiency of data intensive computing platforms. Hardware configurations are rigid and cannot be right sized for the characteristics of each application that may be executed on the platform. We introduced a software mechanism to dynamically right size the hardware configuration by modifying the resource allocations made to each of the worker tasks of the jobs. The resource allocation method learns the characteristics of the jobs and determines the appropriate allocations to efficiently utilize the resources. This results in both lower energy consumption as well as higher throughput, and we saw through evaluations that the savings were significant. The changes in resource allocation do affect the latency performance of the jobs and we illustrated this trade-off with an example execution. The proposed methods demonstrate an important opportunity to improve energy efficiency and help reveal several additional challenges and extensions that may help improve data intensive computing infrastructures.

References

[1] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119–128, 2007.

[2] Michael J. Cafarella and Christopher Ré. Manimal: relational optimization for data-intensive programs. In *Procceedings of the 13th International Workshop on the Web and Databases*, WebDB '10, pages 10:1–10:6, 2010.

[3] Rong Chen, Haibo Chen, and Binyu Zang. Tiled-mapreduce: optimizing resource usages of data-parallel applications on multicore with tiling. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 523–534, 2010.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53:72–77, January 2010.

[5] Apache hadoop. `http://hadoop.apache.org/`.

[6] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50, 2009.

[7] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41:59–72, March 2007.

[8] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 261–276, 2009.

[9] Jacob Leverich and Christos Kozyrakis. On the energy (in)efficiency of Hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44:61–65, March 2010.

[10] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 1–14, 2008.

[11] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Ulfar Erlingsson, Pradeep Kumar Gunda, Jon Currey, Frank McSherry, and Kannan Achan. Some sample programs written in dryadlinq. Technical Report MSR-TR-2009-182, Microsoft Research, December 2009.

[12] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.