Mobile and Embedded Databases

Anil K. Nori
Microsoft Corporation
One Microsoft Way, Redmond, WA 98052
anilnori@microsoft.com

Abstract

Recent advances in device technology and connectivity have paved the way for next generation applications that are data-driven, where data can reside anywhere, can be accessed at any time, from any client. Also, advances in memory technology are increasing the capacities of RAM and Flash, and their costs down. These trends lead to applications that are mobile, embedded, and data-centric. This paper presents an overview of mobile and embedded database systems and applications.

1 The New Environment - Mobile and Embedded

Recent advances in processors, memory, storage, and connectivity have paved the way for next generation applications that are data-driven, whose data can reside anywhere (i.e. on the server, desktop, devices, embedded in applications) and that support access from anywhere (i.e. local, remote, over the network, in connected and disconnected fashion). Memory sizes have gone up and prices have come down significantly; with 64 bit addressability, it is not uncommon to configure servers with 8 – 16GB of memory, and desktops with 2 – 4 GBs of memory. With advances in flash memory technology, large flash drives are available at reasonable prices. Computers with 32 GB flash drives are making way into the market. Flash drives not only eliminate seek time and rotational latency they consume significantly less power than conventional disk drives, making them ideal for portable devices. All this naturally leads to mobile, disconnected, and specialized application architectures. Application components can run on different tiers, in different service (autonomy) boundaries, and on different platforms (e.g. server, desktop, mobile). These new breeds of applications fall into one or more of the following categories:

1.1 Mobile

As more users adopt Wi-Fi enabled laptops, and with increasingly capable mobile devices, the need for mobile applications is increasing. Applications like Email, Calendaring, CRM (Customer Relationship Management) are already targeting mobile devices. Middleware infrastructures like application servers and workflow services are becoming mobile-aware. Some reasons for such mobility trends are:

- More employees are mobile. Email and offline access is becoming pervasive.
- Mobile usage is broadening. Mobile usage is already prevalent in certain vertical domains like Healthcare, Insurance, and Field Services.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- Mobile applications are more than just browser windows – more and more applications now run natively on mobile devices.

Data management and access on mobile devices is central to mobile applications. As mobile applications achieve widespread adoption in the enterprise, mobile and embedded DBMSs – needed to support such applications become an important part of the IT infrastructure. And as these applications grow more disconnected and sophisticated with increasing data sizes the need for rich data processing capabilities increases.

We also note that consumer and home user applications are becoming web-centric and data is being stored on the web (cloud) and accessed from anywhere, at any time. Even data that was traditionally stored in PCs is migrating to the web (cloud), thereby unlocking the data access from a specific location. Mobile devices complete the anywhere data access vision – they provide access from anywhere. Smart mobile devices combine multiple functions of phones, media players, PCs, etc. Such devices are becoming powerful in their processing power and provide larger storage capacity. These advances provide data access and also enable caching of data (from original sources) that can be processed offline

1.2 Streaming

In certain scenarios, data is simply streamed intelligently through application logic. There is no additional need to store the data – except transiently for the duration of the operation. Conventional database systems require data to be first loaded into the database; then the operation is performed, and the data may be later removed from the database. This adds significant complexity to the application, and dramatically reduces its performance and throughput.

Consider, for example, RFID data in applications. RFID tags are portable sensors that communicate over specialized protocols with RFID reader devices. An increasing number of applications have begun to utilize RFID technology including Manufacturing (e.g. Vendor Managed Inventory, Assembly scheduling), Distribution (e.g. Goods delivery, Shipping verification), Retail (e.g. Shelf stocking, Receiving, Store Replenishment and Ordering, Theft, Merchandise Location), Healthcare Industry to identify patients, Tracking books and automated checkout in libraries, etc. In all these applications, RFID readers (devices) generate events when they identify RFID tags. The RFID event streams are filtered, aggregated, transformed, and correlated so that the events can be monitored in real-time. For example, when a truck carrying packages of product parts (with RFID tags) enters (or leaves) a warehouse, the readers generate events. Spurious events are filtered; related products are aggregated; the event data is transformed and presented on a monitoring dashboard in real-time. The event processing is data-centric and typically requires an in-memory rules engine and query processing.

1.3 Disconnected

Distributed and disconnected application architectures fundamentally change the way applications access and manage data. Instead of locking data in a central location, data is moved closer to the applications, and this enables data processing to be performed in an autonomous and efficient fashion. These applications may run in the mid-tier, on desktops, or on mobile devices. Such an environment is inherently disconnected – there is no need for continuous connectivity between the data sources. Data may be accessed from its original sources, transformed and cached (or stored) close to the applications.

For example, consider a product catalog application aggregating product information across multiple back-end application and data sources. Operations against such an aggregate catalog require them to be decomposed into operations on the underlying sources. After the underlying operations are invoked, responses are collected, and results are aggregated into cohesive responses to the end users and businesses. A typical Catalog Browse operation iterates over a large amount of product data, filters it, personalizes it, and then presents the selected data to the users. These operations are data-centric i.e. they require accessing and querying backend data. Accessing large sets of backend data for every catalog operation can be prohibitively expensive, and can significantly impact the response time and throughput. Caching and querying is therefore a key mechanism in application servers for performance and functionality.

1.4 Embedded

Many applications perform simple to moderate manipulation of data. They need a way of storing, retrieving and manipulating data within the application. These applications themselves are not complex in nature, and are designed to meet a specific user need. Typically, they are developed by vendors specializing in industry verticals (domains) – e.g. Healthcare, Finance etc. These vendors are domain experts but not database specialists. Their focus is the application and would rather not spend their time in database system installation, deployment, and management. While synchronization with backend databases is important, this is typically never seen by the application developer. The databases are typically single application databases; the applications do not want to share (or coordinate) their database with others (which is the case with general purpose database services). These applications could run on devices, desktops, or servers. The ideal way of deployment is deploying the database system components along with the application as a single install. Additionally, different applications may have varying needs from the database system – some may require only ISAM access, while others may require general query support; still others may require synchronization with backend database systems.

While the vertical (embedded) application domain is rapidly growing, traditional DBMS vendors have not paid attention to their needs. Therefore, application vendors have used home grown components for data management using technologies they are familiar with. Files, XML, a rudimentary store and retrieve mechanism, or custom data management implementations are some techniques employed. Consequently, application developers are looking to DBMS vendors for better support for embedded databases to enable their scenarios

It is important to note that new environments and applications span hardware tiers – from devices to desktops to servers and clusters and must work in all tiers.

2 Device Trends

Devices, as relevant to this paper, can be categorized into Mobile devices and Sensor devices. Mobile devices are operated by an end-user to access personal and business data; sensor devices are self-operating and typically collect data from their surrounding environment and provide access to the collected data. Both these devices and their operations impact mobile and embedded database system design and usage. In this section we provide an overview of the advances in these device technologies.

2.1 Mobile Devices

Phones (including smart phones) constitute majority of the mobile devices of interest. A simple phone includes a CPU, memory, storage (flash), networking, and a battery. More recently, phones started offering additional features like camera, music player, GPS, etc. Such convergence of capabilities has lead way to System-on-Chip (SOC) design, with a single chip including a processor, memory, connectivity, camera, music player, video broadcast, GPS, 3D graphics, etc. Smart phones are beginning to run on multi-core processors. Power consumption is still a huge factor in the overall hardware and software system design. Multi core-processors provide asymmetric functionality and can be shutdown and started dynamically as needed to alleviate power consumption. For example, less power consuming operations for multimedia processing can be done on one of the cores. All these advances in mobile phones leads to more data on them, thereby requiring better data management solutions.

2.2 Sensor Devices

Sensor devices are small devices operating with extremely low power consumption (e.g. milliwatts) on batteries. 1000s of these devices are connected, in a geographic network, to form a Sensor Network. Sensor networks are used to collect physical measurements like temperature, humidity, light, etc. in remote environments. A typical sensor device consists of four basic components: sensing, processing, transceiver, and power units. These remote devices are also known as motes. Each mote (e.g. Mica2) contains 4 KB memory, 128 KB – 512 KB flash, with

900MHz multi-channel transceiver. The Mica2 mote weighs about 18 grams and operates on 2 AA batteries lasting for one year.

Each device in the sensor network, known as sensor', is static in one location, gathers data from its surroundings, and streams the data to a central site. Sensors are self-managed (no human interaction) and collect data continuously. They are capable of doing computation and can perform data filtering. Very small footprint database systems (like TinyDB) can run on these mote devices. The federation of databases across the sensor network can send data to a central location in response to a query. In fact, TinyDB supports a variant of SQL as the query language.

2.3 Device Storage Trends

Flash is the primary storage media in both mobile (e.g. phones) and sensor devices. Flash is non-volatile, provides reasonable sizes at affordable prices, and significant advances are taking place in its storage. The capacities are increasing and prices are coming down. The combination of low cost and high capacity are making flash drives very competitive relative to hard disks. In this section we describe some of the key trends in flash storage.
Increase in storage size – Most mobile phones use 1 – 2 GB flash disks today. More applications (like mail, address book, music, pictures) are beginning to take advantage of flash storage. There are also laptop configurations that come with 32 GB flash (currently they are expensive). During the ten years from 1995, flash capacities have doubled every year, growing from 16Mb to 16Gb. By 2012, the capacity is expected to reach 1 Tb. By packaging multiple flash chips, 1 – 2TB disks can be expected in 5 years.
Costs coming down – Today, an 8 Gb (1 GB) flash chip costs about $5. By 2012, the cost is expected to go down 10 times – a 1 Tb (or 128 GB) is expected to cost about $50. That is, a 1 TB flash disk will cost about $400 in 2012. Flash cost is 50X of disks today but expected to be 10X of disks by 2012. Even though Flash is expensive compared to disks, it provides better power utilization and high I/O operations and rates.
Higher IOPS (I/O Operations per Second) – Unlike hard disks, flash storage does not have seek or rotational latencies. Therefore, it can offer higher IOPS for random reads and writes. Current manufacturers claim 10X IOPS compared with hard disks. However, with advances in technology, flash disks with close to 3000 IOPS (for random read/writes) are becoming a reality.
Power consumption – Flash disks consume less power than hard disks, typically in the milliwatt range. Hard disks consume 10 – 15 watts depending on the rotational speed of the disks. Due to the low power consumption, flash is the ideal storage media for sensor devices and mobile phones.
Read/Write characteristics – Flash's read/write mechanism has impact on the database storage design using flash. The read and write operations happen at page granularity. Pages are organized into blocks, typically of 32 or 64 pages. A page can only be written after erasing the entire block to which the page belongs. Page write cost is typically higher than read, and the block erase requirement makes writes even more expensive. A block wears out after 10,000 to 100,000 repeated writes, and so write load should be spread out evenly across the chip. These storage trends warrant some redesign of the storage engine technology in the current mobile and embedded database systems.

These positive trends in device storage are leading way to more data being stored on them that require better data access and management.

3 Mobile and Embedded Applications

In this section, we provide characteristics and examples of mobile and embedded applications.

3.1 Mobile Applications

As employee mobility is increasing, so is the need for traditional desktop applications to run on mobile devices. In the Consumer and Information Worker space, E-mail, Address Book, and Calendaring are three widely used mobile applications. However, these applications are still very simple. As device hardware (processing and

memory capacity) advances, users will demand richer capabilities in these applications. For example, consider a rich Calendaring application – with support for checking and scheduling appointments and meetings, sharing calendars across collaborating workers, integrating calendars with other applications, and so on. In the enterprise space, mobile sales personnel will require CRM applications running on their mobile devices; field service employees will need the ability to check product specifications and perform on-line ordering from mobile devices. Following is a list of some representative mobile application scenarios. These are real examples taken from Microsoft's SQL Server Compact Edition customer scenarios, but apply to any mobile DBMS.

Route delivery management – Drivers get route data on a daily basis that is synchronized when they dock their mobile devices. Mobile DMBS provides the local data store on the devices and the data is synchronized to a backend data source

Utilities consumption reading – The solution provides an end to end capability for reading of Oil, Water, Gas and Electricity meters. Field staff use Pocket PC devices to capture meter readings and companies are interested in making the application available through smart phones also.

Mobile CRM – Mobile CRM provides SFA and CRM solution on the devices. The solution typically integrates into other ERP applications. The DBMS provides the local data store and data synchronization (replication) mechanisms. The replication mechanisms work over a variety of transports (e.g. WiFi, Bluetooth, GPRS, 3G, etc).

Sensor Databases – Data collected by the sensor devices is stored in the local DBMS on the device. Such mobile DBMS systems must operate on extremely constrained configurations (e.g. low power, small memory, NVRAM). The sensor devices are typically placed in remote locations (to study remote environments) and monitored from a central site. Such monitoring requires data from individual DBMSs to be queried and aggregated. The network of sensor DBMSs forms a sensor network of federated DBMSs that is queryable from the central site.

3.2 Embedded Applications

The characteristics of embedded applications are described in section 1. Most mobile applications are embedded applications and typically mid-tier applications are embedded and embed a database system (cache) for performance and manageability. Also, most low-end applications are embedded – e.g. Microsoft Access. These applications are self-managed, self-hosted, and very portable. They are developed using simple-to-use developer tools and are also used as offline/local applications. Following are some examples of embedded database applications.

Desktop Media applications – Windows Vista integrates home entertainment into the PC. It delivers an easy and powerful way to manage digital entertainment – photos, music, TV, movies, videos, radio, etc. The SQL CE DBMS is used as an embedded database system for storing this media data e.g. TV listing information is stored and queried using regular SQL; Media Player Playlists and Ratings are stored for efficient organization and query; Photo Organization data is stored for flexible organization and ad-hoc query.

Line of Business applications – Typical LOB applications are multi-tier applications where data in the back-end data source tends to be authoritative. Data is cached in the middle-tier as reference data and application logic executes over it. This reference data is typically integrated from multiple backend data/application sources, transformed into a format suitable for application logic to process efficiently, and brought close to the application in the mid-tier. Also, the reference data is often read-only and suitable for caching within the application. Embedded Database systems are used for such reference data caching and they provide rich storage management and query on the cached data. As explained earlier, embedded DBMSs also make application packaging and installation easier. For similar reasons, embedded database systems are also used as local caches or stores in client applications. Another important scenario in LOB applications is offline experience. Consider a sales person who routinely maintains her customer relationships. The sales person, even while not connected to the corporate network, must be able to create new customers, update customer information, and gather requirements. This is

an example of the mobile CRM scenario which requires the application to be executed while the salesperson is mobile and offline. It should also be noted that even web application architectures follow similar application and data processing patterns, requiring the need for embedded database systems in the web application tier and even in the browser. For example, recently Google released a browser extension (Google Gears) that embeds SQLite to provide offline capabilities to browser applications.

Stream processing – In section 1, we described the streaming data environment. Stream processing is different from data processing of traditional relational database systems. In stream processing engines, data is processed as it arrives and before it is stored. In-memory embedded DBMS systems can be used in such stream processing engines.

4 Mobile and Embedded DBMS Characteristics

The data access and management requirements of the applications described above are significantly different from that of traditional server DBMSs. These new applications must be able to run on multiple tiers ranging from devices to servers to web and would benefit from various existing database mechanisms. However, these database mechanisms (like query, indexing, persistence) must be unlocked from the traditional monolithic DBMSs and made available as embeddable components (e.g. DLLs) that can be embedded within applications, thereby, enabling them to meet the requirements described above. Such Mobile and Embedded DBMSs have the following characteristics:

1. **Embeddable in applications** – Mobile and Embedded DBMSs form an integral part of the application or the application infrastructure, often requiring no administration. Database functionality is delivered as part of the application (or app infrastructure). While the database must be embeddable as a DLL in applications, it must also be possible to deploy it as a stand-alone DBMS with support for multiple transactions and applications.
2. **Small footprint** – For many applications, especially those that are downloadable, it is important to minimize DBMS footprint. Since the database system is part of the application, the size of the DBMS affects the overall application footprint. In addition to the small footprint, it is also desirable to have short code paths for efficient application execution. Most of these applications do not require the full functionality of commercial DBMSs; they require simple query and execute in constrained environments.
3. **Run on mobile devices** – The DBMSs that run on mobile devices tend to be specialized versions of mobile and embedded DBMSs. In addition to handling the memory, disk and processor limitations of these devices, the DBMS must also run on specialized operating systems. The DBMS must be able to store and forward data to the back-end databases as synchronization with backend systems is critical for them.
4. **Componentized DBMS** – Often, to support the small footprint requirement, it is important to include only the functionality that is required by the applications. For example, many simple applications just require ISAM like record-oriented access. For these applications, there is no need to include the query processor, thereby increasing the footprint. Similarly, many mobile and mid-tier applications require only a small set of relational operators while others require XML access and not relational access. So, it should be possible to pick and choose the desired components.
5. **Self managed DBMS** – The embedded DBMS is invisible to the application user. There can be no DBA to manage the database and operations like backups, recovery, indexing, tuning etc. cannot be initiated by a DBA. If the database crashes, the recovery must start instantaneously. The database must be self managed or managed by the application. Also, embedded DBMS must auto install with the application – it should not be installed explicitly (user action) or independently. Similarly when the application is shut down, the DBMS must transparently shutdown.

6. **In-Memory DBMS** – These are specialized DBMSs serving applications that require high performance on data that is small enough to be contained in main memory. In-memory DBMSs require specialized query processing and indexing techniques that are optimized for main memory usage. Such DBMSs also can support data that may never get persisted.
7. **Portable databases** – There are many applications which require very simple deployment – installing the application should install the database associated with it. This requires the database to be highly portable. Typically, single file databases (e.g. like Microsoft Access databases) are ideally suited for this purpose. Again, there should be no need to install the DBMS separately – installing the application installs the DBMS and then copying the database file completes the application migration.
8. **No code in the database** – Portable database must also be safe. Executable code can be a carrier of virus or other threats. By eliminating any code storage in the database, it can be made safer and portable.
9. **Synchronize with back-end data sources** – In the case of mobile and cached scenarios, it must be possible to synchronize the data with the back-end data sources. In typical mid-tier (application server) caches, the data is fetched from the back-end databases into the cache, operated on, and synchronized with the back-end database.
10. **Remote management** – While mobile and embedded DBMSs must be self managed, it is important to allow them to be managed remotely also, especially those on mobile devices. In enterprises (e.g. FedEX, UPS), mobile devices must be configured and managed in a manner compliant with the company standards. Therefore centralized remote management of these devices is necessary.
11. **Custom programming interfaces** – An important usage of embedded DBMS is in specialized data-centric applications. Such applications use variety of data models (e.g. Relational, XML, Streams, and Analytics) and query languages. The embedded DBMSs must be componentized and extensible to allow (application) domain-specific query languages and programming surfaces.

4.1 Mobile vs. Embedded DBMS

While both mobile and embedded DBMSs share many common characteristics, there are also differences that separate them, especially in deployment. In fact, mobile DBMSs are typically embedded DBMSs but considerably constrained by the environment in which they must execute and perform. The following table illustrates key differences between the two:

Mobile DBMS	**Embedded DBMS**
Targets device tier. Supports device scenarios	Targets all tiers. Deployment is application-specific
Constrained by device physical characteristics	Constrained by the deployment environment
Power, memory size impact the design	Power and media are not constraints
Size (small footprint) is critical	Small size is important
Componentization is not critical but helps minimize size	Componentization is critical to support varied deployments
Scale and throughput are not critical	Scale and throughput are important

5 Mobile and Embedded DBMS Design Considerations

While the architecture of mobile and embedded DBMSs is similar to that of traditional relational DBMSs, the characteristics described in the previous section must be factored in. Some of these characteristics are more critical than others – componentization, small footprint, and self-management are by far the most critical characteristics. In fact, good componentized DBMS architecture naturally makes way for other characteristics like embeddability, size, deployment, etc. This section describes the key design considerations for addressing the mobile and embedded DBMS requirements.

5.1 Componentization

The components of a mobile and embedded DBMS are not really new but how well the functionality is factored and layered within and across the components is important. The key high level components are: Storage Engine, Query Processor, Data Access APIs, and Synchronization. Since specialized database systems and embedded applications know the specific database functionality they desire, it must be possible to choose the specific components and functionality. For these applications one size does not fit all. Custom application-specific packaging offers both performance and size benefits. If an embedded application is single-threaded and does not require isolation, lock management can be factored out. Similarly, if an application does not require JOINs, the join code can be factored out from the query processor – thereby reducing the overall size.

Componentization also provides extensibility. For example, consider processing of structured (Relational) and semi-structured (XML) data with SQL and XQuery respectively. In implementing DBMSs for such support, we believe a common storage engine component can be used with two query processing components, one for SQL and the other for XQuery. Such an architecture not only allows for reuse of components, it also allows for extensibility for plugging additional application domain-specific query processors.

Architecturally, a well factored DBMSs forms an inverted triangle of components, with one (or very small number of) storage engines at the bottom and multiple query execution engines, query optimizers, query compilers, APIs layers, and language bindings at the top.

In the next few sections we look at sub-componentization within these higher level components.

5.2 Storage Engine

Most storage engines support media management to record/row management with transactions, recovery, and efficiency. The storage engine can be componentized as follows:

- Media management (disk, flash, and memory) – While most mobile and embedded DBMS's storage engines must support data on disks, they are also embedded in applications whose data is primarily memory-resident. The storage engine must turn off persistence to disk and optimize large memory use. Mobile DBMS storage engines must support flash media, when they are used in mobile and sensor devices. Supporting different media requires different storage and access mechanisms. The storage engine must be extensible to support these mechanisms.
- Transactions – Embedded DBMS must be capable of supporting concurrency control (e.g. locking) and transactions. However, not all embedded applications require the full ACID properties. Most applications require atomicity but the other properties can be optional. For example, a single threaded application does not require isolation; a pure main-memory based application does not require persistence (and durability). Most read-only caches do not even require atomicity; that is, transaction support is optional. In such cases, the storage engine functionality must be componentized so that these applications that don't require transactions can either bypass the transaction management or factor it out of their application. Also, when embedded DBMSs are used as application caches, where the authoritative data comes from backend data sources, data versioning and multi-versioned concurrency control can improve the overall cache performance. However, adding any mechanism can increase the overall footprint (size) but proper componentization allows the application to choose the desired components.
- Access methods – Since embedded DBMSs are used in variety of application scenarios in different storage environments (e.g. with large memory, flash, disk), the storage and access methods must be optimized to take advantage of this environment. For example, in in-memory BI systems, column based storage and compression is desirable for high performance and memory utilization. Similarly hash based access methods are more appropriate for key based access in large memory environments. Broadly, embedded DBMSs must support flexible row formats and multiple access methods (B-Tree, Hash, and Heap).

5.3 Query Processor

The query language requirements tend to be driven by the embedded applications. LOB applications typically require SQL (or some variation) support; document applications may require XPath or XQuery; analytics applications require multi-dimension query; streaming applications may require some form of stream SQL; and so on. Since there is no single query language that can support all embedded applications, either query processors should be optional in the embedded DBMSs or their architecture should be extensible so that multiple query processors can be plugged-in. The query processor itself can be componentized as follows:

- Small number of query execution operators – for example, support for extended relational, analytics operators. The query execution operator interfaces must be exposed so that multiple query languages can be supported on them.
- Multiple query compilers – typical query compiler includes the language processor and the optimizer. These query compilers optimize the queries and generate an execution plan in terms of the query execution operators described above. Again, such an architecture allows for reuse (of execution operators) and extensibility.
- Both the query execution and the query compiler components must be configurable so that applications can pick and choose the desired query functionality.

5.4 Synchronization

Synchronization mechanisms replicate or synchronize data between the application and the backend data sources. Synchronization is important for mobile and embedded applications for the following reasons:

- Mobile and embedded DBMSs are predominantly deployed in the client-tier and mid-tier.
- Data is copied or replicated from the backend data sources close to the applications (e.g. as caches) in the client and middle tier, for better performance and ease of application management.
- Occasional disconnection between the application and the backend data sources causes data to be off-lined into the client databases. Offline databases must be periodically synchronized with the original backend data sources.
- Synchronization must work over variety of communication networks (e.g. LAN, WAN, wireless networks, etc), and to variety of data sources (e.g. database systems, application systems, web services).

5.5 Management

For most mobile and embedded applications, database management is transparent. That is the embedded DBMS is self-managed. In traditional DMBSs, DBAs perform the following management functions but in embedded DBMSs these are done transparently:

- Backup and restore – Unlike in traditional DBMSs, in mobile and embedded DBMSs the backup operation is very simple and is done transparently. Applications specify a policy (typically, through an API) that will schedule periodic but consistent copying of the database files. It is highly desirable to have a single file database so that consistent copying becomes easier. Some embedded DBMSs do not even require a separate log. Restore is essentially copying the consistent copy of the file to the desired location. In systems which have logs, it is necessary to copy the log, along with the database, in a consistent manner without the application being aware of the different files.
- Recovery – Again, the application is not aware of recovery. It is done transparently and immediately as soon as failure is detected. For example, if a transaction is aborted, undo is done transparently. If the system fails, then on restart, the database must be recovered automatically. In log-less systems (i.e. systems where there is no in-place writing), this operation is trivial. In log based systems, the log must be applied transparently.

- Table and index reorganizations – In most mobile and embedded applications, the database workload (query and updates) are known at application development time. Therefore, table and index creations are typically done explicitly. These applications are well tuned before they are deployed. Therefore, dynamic index creation is not a requirement. However, table and index storage re-organization (e.g. compaction) is done automatically, typically at the database level to minimize fragmentation.

6 Conclusions

With the increased interest in specialized applications and database systems (like streaming, analytics, mobile, sensor networks, etc.), the need for Mobile and Embedded DBMSs is increasing. There have been mobile and embedded DBMS products in the market that satisfy some of the characteristics described above, but were designed as low-end database products and do not factor in the recent hardware and application trends. For example, products that focused on small footprint but did not design with componentization for embedded applications; similarly, there are products that only provide embedded (well componentized) storage engines but without any query processing components.

The hardware trends in multi core CPUs, large main memory, and NVRAMs (e.g. flash) are beginning to disrupt the traditional database architectures. Particularly, advances in large memory and flash storage open doors for further research in database architectures. The componentized mobile and embedded DBMSs are in a position to adapt to these changing hardware and software trends more rapidly.

References

[1] *http://research.microsoft.com/~Gray/talks/Flash_is_Good.ppt.*

[2] Michael Stonebraker and Ugur Cetintemel. *One Size Fits All: An Idea Whose Time Has Come and Gone*.

[3] Michael Olson, Keith Bostic, Margo Seltzer. Berkeley DB. *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*.

[4] Margo Seltzer. There is more to data access than SQL. In *ACM Queue, Databases, Vol. 3 No. 3 - April 2005*.

[5] *SQL Server CE: http://www.microsoft.com/sql/editions/compact/default.mspx*.

[6] Suman Nath and Aman Kansai. Dynamic Self-tunig Database for NAND Flash. *ISPN '07, April 25 - 27, 2007, Cambridge, Massachusetts, USA.*

[7] *TinyDB: http://telegraph.cs.berkeley.edu/tinydb/.*

[8] *Window CE Platform, Microsoft Corporation*.