Technical Report No. 2011-581

Managing Data-Intensive Workloads in a Cloud[1,2]

Ph.D. Depth Paper

Student Name: Rizwan Mian

Student Number: 6001533

Supervisor: Dr. Patrick Martin

Supervisory Committee: Dr. Selim Akl and Dr. Ahmed Hassan

School of Computing

Queen's University

Kingston, Ontario, Canada

September 20, 2011

[1] Publications: a book chapter [Mian et al. '11] and a poster [Mian et al. '10] have been published based on this depth paper.

[2] Acknowledgements: this paper has been written under the supervision of Dr. Patrick Martin. Some minor contributions have been made by Andrew Brown and Mingyi Zhang. Dr. Masroor Hussain and Wendy Powley has reviewed a draft of the depth paper. Dr. Hussain has provided detailed feedback. Muhammad Aboelfotoh has proofread the final draft. I thank them all.

ABSTRACT

In current information technology era, the pace and volume of data being generated is exceeding our ability to manage and analyse it. Until recently, large data was managed in bulky repositories and analysed on high-end servers or compute clusters. However, the growing volume of data questions the continuation of this approach for many reasons such as feasibility and affordability. This requires exploring new technologies and methods to swiftly and reliably transform raw data into tangible information and hence knowledge. The promise of "infinite" resources given by the cloud computing paradigm has led to recent interest in exploiting clouds for large-scale data-intensive computing. Given this supposedly infinite resource set, we need a management function that regulates application workload on these resources. Workload management, an important component of systems management, is the discipline of effectively managing, controlling and monitoring application workload across computing systems. Meanwhile, data-intensive computing presents new challenges for systems management in the cloud including diverse data processing frameworks, such as MapReduce and Dataflow-processing, and costs inherent with large data sets in distributed environments. We need to establish our current location in this landscape. Therefore, we examine the state-of-the-art of workload management for data-intensive computing in clouds. A taxonomy is presented for workload management of data-intensive computing in the cloud. We use the taxonomy to classify and evaluate current workload management mechanisms and systems.

Keywords: data-intensive workload, workload management, cloud computing, taxonomy, survey

1. Introduction

The advances in information and communication technologies in the last few decades has resulted in an increasingly perceived vision that computing would be available as a utility [Buyya *et al.* '09]. A number of paradigms have been explored to offer computing as a utility, the latest one being *cloud computing*. Due to its recent arrival there has been some confusion on its definition, and there have been many suggestions [Vaquero *et al.* '08]. We like the one proposed by Foster *et al.* (08):

> "*A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers [users] over the Internet."*

The keywords in this definition, such as "dynamically-scalable" or "virtualized", may seem like a marketing mantra. In reality, the keywords are necessary to differentiate cloud computing from typical computing clusters (which usually have a static size), computing grids (typically non-virtual resources with multiple administrative domains), typical web hosting, and in-house or external datacenters (requiring some upfront commitments or capital costs).

Cloud computing is being driven by economic and technological factors and companies such as Amazon, IBM, Microsoft and Google are providing software and computing resources as services [Amazon 'a;

Chappell '09; Google ; IBM]. Cloud computing provides users the "illusion" of infinite resources [Armbrust *et al.* '10; Armbrust *et al.* '09], available on demand while providing efficiencies for application providers by limiting up-front capital expenses and by reducing the cost of ownership over time.

Cloud computing is helping in realizing the potential of large-scale data-intensive computing by providing effective scaling of resources. A growing number of companies, for example Amazon and Google, rely on their ability to process large volumes of data to drive their core business [Dean *et al.* '08]. On the other hand, the scientific community is also benefiting in application areas such as astronomy [Raicu *et al.* '06] and life sciences [Desprez *et al.* '06] that have very large data sets to store and process. Figure 1 gives a high-level conceptual view of a cloud.

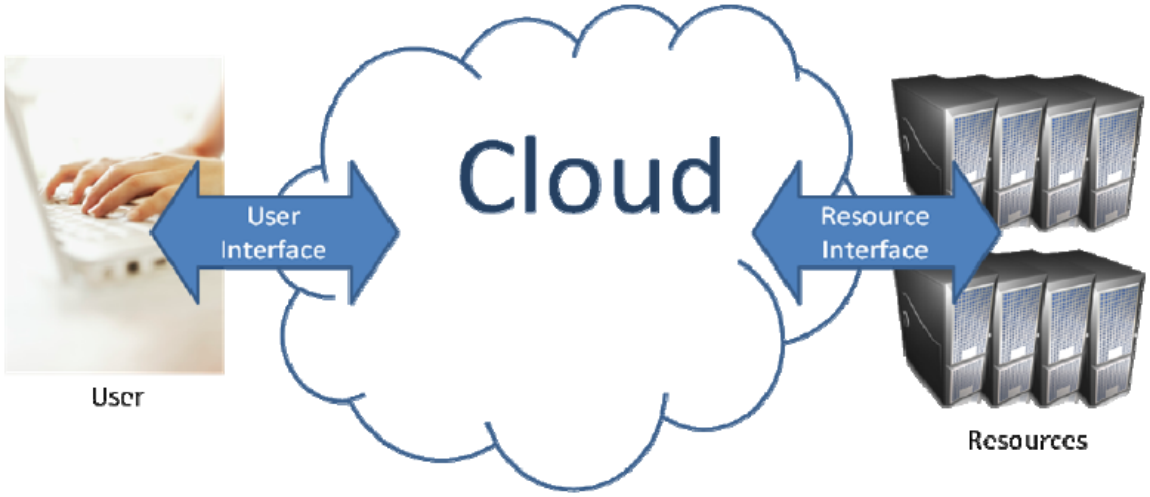

Figure 1: high-level view of a cloud [Bégin '08].

Data-intensive computing poses new challenges for systems management in the cloud. One challenge is that data-intensive applications may be built upon conventional frameworks, such as shared-nothing parallel database management systems [Dewitt *et al.* '92], or modern frameworks, such as MapReduce [Dean *et al.* '04], and so have very different resource requirements. A second challenge is that the parallel nature of large-scale data-intensive applications requires that scheduling and resource allocation be done to avoid data transfer bottlenecks. A third challenge is to support effective scaling of resources when large volumes of data are involved.

Workload management is the discipline of effectively managing, controlling and monitoring application workloads across computing systems [Niu *et al.* '09]. In a cloud, the two main mechanisms used for workload management are (a) scheduling requests, and (b) provisioning resources. Since the load on data resources in a cloud can fluctuate rapidly among its multiple workloads, it is impossible for systems administrators to manually adjust the system configurations in order to maintain the workloads' objectives during their execution. Therefore, we need the ability to automatically manage the workloads on data resources.

The primary objective of this paper is to provide a systematic study of workload management in today's clouds by surveying the workload management systems and techniques implemented. In this report, we propose a taxonomy to classify workload management techniques and evaluate today's workload management systems in cloud computing.

1.1. What's new in cloud computing?

Distributed, loosely-coupled clusters have been available for quite some time and there is a large amount of middleware available for such clusters [Henderson '95; Zhou *et al.* '93; Litzkow *et al.* '88]; so what is new with clouds?

The first new thing is about the scale of resources. Google and Yahoo have used clouds that contain thousands of loosely coupled commodity PCs [Grossman *et al.* '09]. With Hadoop [Apache 'a] (an open source MapReduce implementation), terabytes of data can be processed easily, something that requires significant overhead with a database. The overhead includes developing relational schemas and populating tables.

The second new thing is the simplicity of access to resources that clouds provide. To substantiate this claim, Grossman *et al.* (09) provide some examples:

- With a MapReduce system, a novice software developer can be analyzing terabytes of web data on 100 nodes by developing a small MapReduce programs in a short period of time that might even be less than a day. This is the strength of MapReduce platform.

- With just a credit card and a connection to the Internet, it is possible to use Amazon's Elastic Compute Cloud (EC2) [Amazon 'a] to perform a computation on 100 processors without any capital investment, without the help of a system administrator, and without installing or managing any middleware.

The third new thing is the *elasticity* of resources, which is the ability to change the number of resources during workload execution. For example, it is possible to manually increase the number of processors rented from Amazon EC2 with a few mouse clicks, or automatically using a load balancer.

This style of cloud computing has its origins in industry [Grossman *et al.* '09]. It has evolved to answer industry's needs for a simple to use, yet scalable and robust platform to process large volumes of data.

1.2. Why is cloud computing gaining relevance?

Three aspects from a hardware provisioning and pricing point of view are fuelling the popularity of cloud computing [Armbrust *et al.* '09]:

1. The "illusion" of infinite computing resources available on demand, quickly enough to follow load surges. As a result, cloud users need not to plan far ahead for ensuing availability of resources.

2. The elimination of an up-front commitment by cloud users, thereby allowing companies to follow an organic growth: start small and increase hardware resources only when there is an increase in their needs.

3. The ability to use computing resources on a short-term basis (for example, processors by the hour and storage by the day) and paying for them on a pay-as-you-go basis, thereby rewarding the release of machines and storage when they are no longer useful.

Efforts in the past at providing utility computing failed, and Armbrust *et al.* (09) note that in each case one or two of these three critical characteristics were missing. For example, Intel Computing Services in 2000-2001 required negotiating a contract for longer-term use than per hour. For many companies, the pay-as-you-go cloud computing model, along with having someone else looking after the hardware, is very attractive [Abadi '09]. There arises a question, while the attraction to cloud computing users is clear, who would become a cloud computing provider and why?

To build, provision, and launch a large datacenter for a cloud is a multi-million dollar investment. Therefore, constructing and operating an extremely large-scale commodity-compute datacenters, at low-cost locations, is necessary to enable cloud computing. Such a datacenter would consist of tens of thousands of resources. This amount of resources enables a 5 to 7 times decrease in cost of electricity, network bandwidth, operations, software, and hardware available because of very large economies of scale. These factors, combined with statistical multiplexing to increase utilization compared to traditional datacenters, means that cloud computing could offer services below the costs of a medium-sized datacenter and yet still make a good profit [Armbrust *et al.* '10].

1.3. What are the opportunities in cloud computing?

Any fundamentally new types of applications enabled by cloud computing are yet to be seen. Nevertheless, several existing applications are likely to benefit greatly from cloud computing, and could contribute to its growth. We discuss two applications suggested by Armbrust *et al.* (09) that are particularly relevant to our study:

1. **Parallel batch processing.** The elasticity in cloud computing presents a unique opportunity for computational batch-processing and analytics jobs that process terabytes of data that would, otherwise, take a long time to complete. Provided that there is enough computational or data parallelism in the application, users can take advantage of the cloud's new "illusion" of infinite resources and balance that against a pay-as-you-go model. "Cost associativity" is relevant here: using hundreds of computers for a short time costs the same as using a few computers for longer time. For example, The Washington Post used 200 Amazon's EC2 resources (1,407 server hours) to convert 17,481 pages of Hillary Clinton's travel documents into a form more friendly to use on the web within nine hours after they were released [Amazon 'l]. Programming abstractions, such as Google's MapReduce [Dean *et al.* '08] and its open-source counterpart Hadoop [Apache 'a], were originally designed to execute over large clusters. They allow programmers to express tasks, such as the conversion task described above, at the domain level while hiding the operational complexity of orchestrating parallel execution across hundreds of cloud computing servers. Although originally designed to execute over large clusters, programming abstractions such as Google's MapReduce [Dean *et al.* '08] and its open-source counterpart Hadoop [Apache 'a] allow programmers to express such tasks while hiding the operational complexity of orchestrating parallel execution across hundreds of cloud computing servers. Amazon has exposed a MapReduce implementation [Amazon 'e] on clouds. A start-up company [Cloudera] is also pursuing commercial opportunities in this space.
2. **Analytics.** The database industry was originally dominated by transaction processing. That demand is now levelling off and instead a growing share of computing resources is now spent on understanding customers, supply chains, buying habits, ranking, and more. Hence, while online transaction volumes will continue to grow slowly, decision support is growing rapidly. Where 1 terabyte is considered large for transactional systems, petabyte is increasingly becoming a norm for analytical systems [Abadi '09]. The real value of data lies in analysis, and analysing at petabyte scale requires large number of resources.

We add two more applications to this set:

1. **Surge Computing.** Some applications can be hosted in an internal datacenter (a *private cloud*) which can scale out to *public clouds* when additional resources are needed. This technique is known as *surge computing*. This combination of private and public clouds is known as a *hybrid cloud*. The reasons for having private clouds include development, testing, or to maintain critical service levels in case of public cloud failure.

2. **Cluster in clouds.** Today, many of the high-performance computing (HPC) tasks are executed on compute clusters, which are often poorly utilized [Armbrust *et al*. '10]. Building a conventional cluster from cloud resources is worth investigating. This would enable transparent execution of legacy cluster applications on clouds with the critical benefit of elasticity. That is to increase cluster resources when an application requires them and to release when the demand tails off.

1.4. What are the obstacles in cloud computing?

Armbrust *et al*. (10) ranked the critical obstacles to the growth of cloud computing as shown in Table 1. The first three affect adoption, the next five affect growth, and the last two are policy and business obstacles. Each obstacle is paired with an opportunity to overcome that obstacle, ranging from product development to research projects.

#	**Obstacle**	**Opportunity**
1	Availability/business continuity	Use multiple cloud providers
2	Data lock-in	Standardize APIs; compatible software to enable surge computing
3	Data confidentiality and auditability	Deploy encryption, VLANs, firewalls
4	Data transfer bottlenecks	FedExing disks; higher bandwidth switches
5	Performance unpredictability	Improved virtual machine support; flash memory; gang schedule virtual machines
6	Scalable storage	Invent scalable store
7	Bugs in large distributed systems	Invent debugger that relies on distributed virtual machines
8	Scaling quickly	Invent auto-scaler that relies on machine learning; snapshots for conservation
9	Reputation fate sharing	Offer reputation-guarding services like those for email
10	Software Licensing	Pay-for-use licenses

Table 1: Top 10 obstacles to and opportunities for growth of cloud computing [Armbrust *et al.* '10].

Note, some of these obstacles (2, 9, 10) are not strictly technical. obstacle (3) has little or nothing to do with the mechanics of enabling large-scale computational or data processing. The business continuity part of obstacle (1) is non-technical. As long as enough resources are available, the availability part of obstacle (1) – though technical – is a non-issue. Locating bugs in large distributed systems (7) becomes part of the application development toolkit and has little to do with the mechanics of large-scale data processing. Addressing these obstacles may result in technical solutions as in the case of suggested opportunities for obstacles (1, 2, 3, 9, and 10).

Nevertheless, three of the identified obstacles (4, 6, 8) closely relate to data. So, data aspects are critical to the successful growth of cloud computing. This makes relevant data research very important.

1.5. What are the offerings of a cloud?

At present, cloud providers offer three service abstractions (depicted in Figure 2): infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS) [Prodan *et al.* '09].

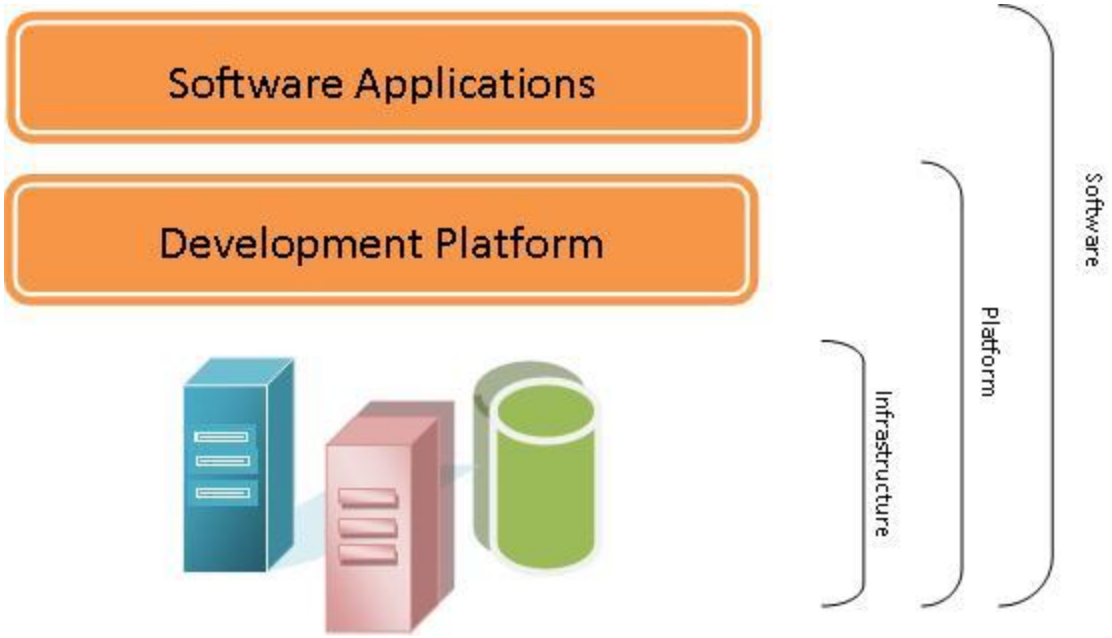

Figure 2: abstractions of cloud: Infrastructure-as-a-Service (Iaas), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) [e2enetworks].

In *IaaS*, the cloud providers expose their infrastructure as a service to the users. For example, Amazon's Elastic Compute Cloud (EC2) [Amazon 'a] exposes its infrastructure in terms of virtual machines. In *PaaS*, software frameworks or platforms are exposed as a service to the users. The users write their software for a specific platform which that provider hosts. After writing software, the users upload their software to the provider's system and run it there whenever needed. Google App Engine [Google] offers this level of abstraction.

The services themselves have long been referred to as Software as a Service (SaaS). In *SaaS*, software is exposed as a service on the web. SaaS is similar in spirit to web services or web applications. Google's online applications – such as Google Calendar – can be thought of as SaaS.

1.6. What is data-intensive?

A data-intensive computing environment consists of applications that produce, manipulate, or analyse data in the range of hundreds of megabytes to multi-terabytes [Moore *et al.* '98]. This observation is dated but there are examples where this is still relevant. For example, life science applications usually manage databases of protein sequence that are in the order of gigabytes [Desprez *et al.* '06]. The data sizes have generally increased rapidly since Moore's claim. The creation of large digital sky surveys [SDSS] presents the astronomy community with tremendous scientific opportunities. However, these astronomy data sets are generally terabytes in size and contain hundreds of millions of objects separated into millions of files [Raicu *et al.* '06]. High-energy physics remains a leading generator of raw data. For example, 3,200 terabytes of data will be seen each year in the Atlas experiment for the Large Hadron Collider (LHC) at the Center for European Nuclear Research [CERN].

The data is organised as collections or *data sets* and are typically stored on mass storage systems such as tape libraries or disk arrays. The data sets are accessed by users in different geographical locations. The

users may create local copies or *replicas* of the data sets to reduce data access latencies to improve application performance. A replica may be a complete or a partial copy of the original data set.

The amount of data growth is intimidating, and the time needed to move it where needed has increased to days. Increased transmission time is a consequence of the divergence of CPU execution growth rates from network bandwidth growth rates [Moore *et al.* '98]. The transmission time has greatly reduced in the local area network with the emergence of high speed communication links such as InfiniBand [Koop *et al.* '08]. Even if the financial cost of such links is set aside, the sheer amount of data shadows the gain provided by the high speed communication links. Moreover, the cost of wide-area networking has fallen more slowly and Gray (08) concludes that economic necessity mandates putting the data near the application. We are generating data more rapidly than we can move, manage or process.

1.7. What is workload management?

We define *workload* as a set of requests that access and process data under some constraints. The data access performed by a request can vary from retrieval of a single record to the scan of an entire file or table. The requests in a workload share a common property or set of properties, such as the same source application or client, type of request, priority, or performance objectives.

On a shared data server, executing workloads compete with each other for system resources such as processors, main memory, disk input/output (I/O), network bandwidth and various queues. If workloads are allowed to compete without any control then some workloads (e.g. an analytical workload) may consume a large amount of shared system resources resulting in other workloads missing their objectives (e.g. deadline). As workload requests presenting on a data resource can fluctuate rapidly among multiple types of workloads, it becomes impossible for administrators to manually adjust the system configurations in order to maintain the workloads' objectives during their execution. Thus, it becomes necessary to automatically control the workloads with different objectives and manage shared system resources.

The primary objective of the paper is to provide a systematic study of workload management of data-intensive workloads in clouds. The contributions of the paper are the following:

- a taxonomy of workload management techniques used in clouds.
- a classification of existing mechanisms for workload management based on the taxonomy.
- a survey of workload management systems in clouds.
- a discussion of possible directions for future research in this area.

The remainder of the paper is structured as follows. The second section gives an overview of workload management in traditional database management systems (DBMSs) and data-intensive computing architectures for clouds. The third section describes the taxonomy for workload management in clouds. The fourth section uses the taxonomy to survey existing systems and techniques to manage data-intensive workloads in clouds. The fifth section summarizes the paper and presents directions for future research.

2. Background

First, we discuss cloud characteristics and suitability of data management applications in clouds. Then, we provide an overview of workload management in traditional DBMSs to highlight the main concepts in workload management. Next, we identify the different architectures that have been proposed to support data-intensive computing in a cloud and discuss their impact on workload management.

2.1. Data Management in the cloud

Distributed resources have been successfully used for compute-intensive applications. There has been a natural desire to do the same for data-intensive applications. One may ask what are the basic barriers for enabling data-intensive distributed computing and how should these be addressed?

2.1.1. Cloud characteristics

To decide which data management applications are best suited for deployment on top of a cloud computing infrastructure, Abadi (09) discusses three characteristics of a cloud computing environment:

1. **Compute power is elastic, but only if workload is parallelizable.** One of the key selling points of cloud computing is its elasticity in the face of changing demands and conditions. Commercial clouds offer large numbers of virtual machines (VMs) but these VMs are only available in a few fixed configurations or types [Quiroz *et al.* '09]. The VM types usually differ in their computational or data storage capacity. So additional computational resources are typically obtained by allocating additional VMs to a task. In general, applications designed to run on top of a *shared-nothing architecture*[1] are well suited for such an environment.

2. **Data is stored at an untrusted host.** In general, moving data off user premises increases the number of potential security risks, and mandates appropriate precautions. Furthermore, the data is physically located in a particular country and is subject to local rules and regulations. Abadi (09) claims that the customer has little choice but to assume the worst that the data may be accessed by a third party without the customer's knowledge. We argue a cloud provider could do better in terms of data privacy by notifying customers what third parties may have access to their data. Moreover, a cloud provider should make an effort to address those risks rather than push the responsibility to users.

3. **Data is replicated, often across large geographic distances.** Data availability and durability is paramount for cloud storage providers, as data loss or unavailability can cause unnecessary service downtime leading to ill-repute of the institution (outages often make the news). Large cloud providers with datacenters spread throughout the world have the ability to provide high levels of fault tolerance by replicating data across large geographic distances.

These characteristics are likely to impact on what applications could be deployed in a cloud. In particular, we discuss data management applications below.

2.1.2. Data management applications in the cloud

The cloud characteristics discussed above have clear implications on the data management application suitable for cloud. Abadi (09) describes the suitability of moving the two largest components of the data management market into the cloud: transactional data management and analytical data management.

2.1.2.1. Transactional data management

These applications typically rely on the ACID (Atomicity, Consistency, Isolation and Durability) guarantees that databases provide, and tend to be fairly write-intensive. At present, transactional data management applications are unlikely to be deployed in the cloud for the following reasons [Abadi '09]:

1. **Transactional data management systems do not typically use a shared-nothing architecture.** Most commercial DBMSs (Oracle, IBM DB2, Microsoft SQL Server and Sybase) dominate the transactional database market. However, only recently some of these vendors have offered shared-nothing

[1] a set of independent machines accomplishing a task with minimal resource overlap.

architecture but only for analytical applications running on data warehouses. One may argue that nothing is stopping these vendors for providing shared-nothing architecture for transactional applications. However, implementing a transactional database system using a shared-nothing architecture is non-trivial, since data is partitioned across sites and, in general, transactions cannot be restricted to accessing data from a single site. Such a system would require complicated distributed locking and commit protocols; the data would be transported over the network leading to increased latency and reduced bandwidth. Furthermore, the overwhelming majority of transactional data processing deployments are less than 1 terabyte in size [Stonebraker *et al.* '07]. This undermines the main benefit of a shared-nothing architecture which is its scalability.

2. **It is hard to maintain ACID guarantees in the face of data replication over large geographic distances.** The CAP theorem [Gilbert *et al.* '02] shows that a shared-data system can only choose at most two out of three properties: *consistency*, *availability*, and tolerance for data partitions (*partition-tolerance*) in case of failures. A system is essentially left with the choice of consistency and availability when data is replicated over a wide area [Abadi '09]. As a consequence, consistency is often compromised to get acceptable system availability.

   Recent systems including Amazon's SimpleDB [Amazon 'k], Yahoo's PNUTS [Brian *et al.* '08] and Google's BigTable [Chang *et al.* '08] relax all or some ACID guarantees. These systems weaken the consistency model by implementing various forms of eventual/timeline consistency mechanisms. In the weak consistency mechanisms, all replicas do not have to agree on the current value of a stored value (avoiding distributed commit protocols), or provide simple read/write store (not implementing general purpose transactions).

3. **There are enormous risks in storing transactional data on an untrusted host.** The complete set of data to support critical business applications typically resides in transactional databases. This data includes fine grain data, which can consist of sensitive information such as bank account details. Any increase in potential security breaches or privacy violations is typically unacceptable. The cloud providers need to explicitly provide certain minimal security guarantees, and it is worth exploring what security sensitive information can be handled with these weaker semantics.

Even if we overlook the security aspect of transactional data, we see that there are difficulties in providing efficient distributed locking and commit protocols that would require minimal data to be transferred over the network. More importantly, strict ACID guarantees cannot be provided over large geographic distances. Thus, the transactional data management applications are generally not well suited for cloud deployment.

2.1.2.2. Analytical data management

Analytical data management refers to applications that query a data store for use in planning, problem solving, and decision support. The typical sources of such analyses include operational and historical data from multiple operational databases and archives. As a result, the scale of analytical systems is generally larger than their counterpart transactional systems. Where 1 terabyte is considered large for transactional systems, petabyte is increasingly becoming a norm for analytical systems [Sybase '07]. Furthermore, analytical systems tend to be read-mostly (or read-only), with occasional batch inserts. Analytical data management systems are well-suited to run in a cloud environment for the following reasons [Abadi '09]:

1. **Shared-nothing architecture is a good match for analytical data management.** The shared-nothing architecture is commonly considered to scale well. The rare writes in analytical databases eliminates the need for complex distributed locking and commit protocols. Therefore, the shared-nothing architecture becomes a promising candidate to manage large amount of data used by analytical workloads. Such workloads typically comprise of many large scans, multidimensional aggregations,

and star schema joins, all of which are fairly easy to parallelize across nodes in a shared-nothing network.

2. **ACID guarantees are typically not needed.** The writes are infrequent in analytical database workloads and analysis on a recent snapshot of the data is usually sufficient. This makes the 'A', 'C', and 'I' (atomicity, consistency, and isolation) of ACID easier to obtain. As a result, the consistency requirements and tradeoffs for distributed data are not as problematic for analytical databases as they are for transactional databases.

3. **Particularly sensitive data can often be left out of the analysis.** In many cases, sensitive data that would be most damaging should it fall in wrong hands can be identified. Such data can be excluded from the analytical data store, or included after anonymizing or encrypting. Second, the analytical workload could work on coarse-grain data instead of the lowest level and most detailed data.

From the above discussion, we can see that the characteristics of the data and workloads of typical analytical data management applications are fairly suitable for cloud deployment. The elasticity of the cloud (both compute and storage) is in line with the shared-nothing architecture, while the security risks can be somewhat managed or reduced. In particular, Abadi (09) expects the cloud to be a preferred deployment platform for (a) data warehouses for medium-sized businesses (especially those that do not currently have data warehouses due to the large up-front capital costs), (b) for sudden or short-term business intelligence requirements (e.g., a retail store analyzing purchasing patterns in the aftermath of a tsunami), and (c) for public-facing data warehouses (for which data security is not an issue).

2.2. Workload management in traditional DBMS

There is an increasing trend of consolidating multiple individual databases onto a single shared data server [Niu *et al.* '09]. This leads to the simultaneous presence of multiple types of workloads on a single data resource. These workloads may include on-line transaction processing (OLTP) workloads and on-line analytical processing (OLAP) workloads. OLTP workloads consist of short and efficient transactions that require small amounts of CPU and disk I/O to complete. Whereas, OLAP workloads are typically longer, more complex and resource-intensive queries that can take hours to complete [Zhang *et al.* '11]. Workloads submitted by different applications or initiated from distinct users may have unique objectives that need to be satisfied.

Niu *et al.* (09) define workload management as the discipline of effectively managing, controlling and monitoring "workflow" across computing systems. In their case, a workflow or a workload is primarily a DBMS workload. With the increasing trend towards server consolidation and more diverse workloads, workload management has become an important requirement on a DBMSs. Niu *et al.* (09) also argue that workload management is necessary so the DBMS can be business-objective oriented, provide efficient various levels of services at fine granularity, and maintain high utilization of underlying resources with low management costs.

A workload management process in DBMSs can involve three kinds of controls (as shown in Figure 3) namely admission, scheduling and execution controls [Krompass *et al.* '09]. *Admission control* determines whether or not newly arriving requests can be admitted into a database system. Admission control avoids increasing the load on the when it is busy. Requests have various costs and resource demands in OLAP and OLTP workloads. The admission decision is made based on admission control policies and the estimated arriving query's cost, which is given by the database query optimizer. Workloads with different priorities can have different admission policies and can be associated with a different set of threshold values. The thresholds may include the upper limits for the estimated resource usage of a query or the estimated execution time of the request. If a request's estimated cost exceeds the threshold, the request may be

queued or rejected. High priority workloads usually have higher (easier) thresholds, thus more high priority requests can be admitted to the system to run.

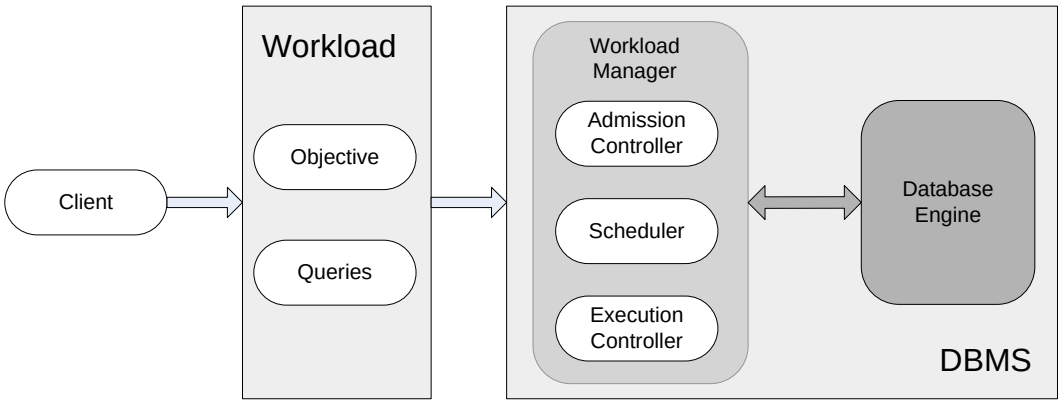

Figure 3: A typical DBMS workload management system includes three components: the admission controller, query scheduler, and execution controller [Krompass et al. '09].

Query scheduling determines when the admitted queries can be sent to the database engine for execution. Its primary goal is to schedule the execution of as many admitted queries as possible while maintaining the system in an optimal state. Admitted queries are scheduled based on scheduling policies and available system capacity. If an admitted query has high priority and a system has available capacity, then the query is scheduled to run immediately in order to meet the required performance levels. The lower priority queries are scheduled in such a way that their execution is tied to the availability of system resources. Traditionally, multi-programming level (MPL), that is the number of requests concurrently running on a database system, is used to manage the system load. If the MPL value is too large, then the system can become overloaded. On the other hand, if the MPL value is too low, then the system may be underutilized and hence system performance suffers. For a database system, different types of workloads have different optimal MPLs [Mehta *et al.* '08]. Scheduling aims to decide how many queries from different types of workloads with different business priorities can be sent to the database engine for execution at the same time.

In contrast with admission and scheduling controls, which are applied to queries before their execution, execution control is imposed during run time. The main goal of *execution control* is to dynamically manage the running processes of large queries in order to limit their impact on other queries by slowing down the queries' execution speed and freeing up shared system resources for use by higher priority queries. As query costs estimated by the database query optimizer may be inaccurate, some long-running and resource-intensive queries might get the chance to enter a system when the system is experiencing a high load. These problematic queries compete with others for the limited available resources and may result in high priority queries getting insufficient resources and missing their performance goals. Execution control manages the running of problematic queries based on execution control policies and determines what queries should be controlled as well as to what degree.

2.3. Data-intensive computing architectures

Data-intensive computing in a cloud involves diverse architectures and workloads, which adds complexity for workload management compared with traditional DBMSs. Workloads in a cloud can range from the ones consisting of relational queries with complex data accesses to the others involving highly parallel MapReduce tasks with simple data accesses. The workloads in clouds can also differ from DBMS workloads with respect to granularity of requests, that is, the amount of data accessed and processed by a

request. To reduce the data movement across the network, data can be processed at the same resource but this reduces parallelism. Increasing parallelism results in increased data movement. We see a tradeoff between amount of (processed) data movement and amount of parallelism. Coarse grain granularity attempts to strike the right balance. It seems that cloud workloads must operate at coarse-grain granularity to be highly scalable to thousands of resources. Cloud workloads are typically coarse-grained in order to localize data access and limit the amount of data movement. In this paper, we consider the following five different architectures for data-intensive computing in the cloud:

1. **MapReduce** is a popular architecture to support parallel processing of large amounts of data on clusters of commodity PCs [Dean *et al.* '08; Dean *et al.* '04]. MapReduce enables expression of simple computations while hiding the details of parallelization, fault-tolerance, data distribution and load balancing from the application developer. A MapReduce computation is composed of two phases, namely the Map and the Reduce phases. Each phase accepts a set of input key/value pairs and produces a set of output key/value pairs. A *map task* takes a set of input pairs and produces sets of key/value pairs grouped by intermediate key values. All pairs with the same intermediate key are passed to the same *reduce task*, which combines these values to form a possibly smaller set of values. Examples of MapReduce systems include Google's implementation [Dean *et al.* '04] and the open-source implementation Hadoop [Apache 'a].

2. **Dataflow-processing** models parallel computations in a two-dimensional graphical form [Gurd *et al.* '85]. Data dependencies between individual nodes are indicated by directed arcs. The nodes represent tasks and encapsulates data processing algorithms, while the edges represent data moving between tasks. Dataflow systems implement this abstract graphical model of computation. Tasks may be custom-made by users or adhere to some formal semantics such as relational queries. Examples of dataflow-processing systems in a cluster include Condor [Thain *et al.* '05], Dryad [Isard *et al.* '07], Clustera [DeWitt *et al.* '08] and Cosmos [Chaiken *et al.* '08].

3. **Shared-nothing relational processing** is dataflow-processing specialized to the relational model and its usage in databases. Parallelism is an unforeseen benefit of the relational model. Relational queries offer many opportunities for fine-grain parallelism since they can be broken into tasks applied to very large collections of data. The dataflow approach to database system design needs a message-based client-server operating system to orchestrate the dataflow between relational operators or tasks executing on data hosts [Dewitt *et al.* '92]. Each task produces a new relation, so the tasks can be composed into highly parallel dataflow graphs or workflows.

   Pipelined parallelism can be achieved by streaming the output of one task into the input of another task. The shared-nothing relational processing moves only processed or a subset of data through the network. However, the benefits of pipelined parallelism are limited because of three factors [Dewitt *et al.* '92]: (1) Relational pipelines are rarely very long – a chain of length ten is unusual; (2) some tasks cannot be pipelined because they do not emit their first output until they have consumed all their inputs, examples include aggregate and sort operators; (3) often, tasks suffer from execution skew. The speedup obtained is limited in such cases.

   Partitioning a relation involves distributing its tuples over several disks. By partitioning the input data among multiple hosts, a relational query can often be split into many tasks each working on its part of the data. Dewitt *et al.* (92) consider it as an ideal situation for speedup and scalability. The partitioned data is the key to partitioned execution. This partitioned data and query execution is called *partitioned parallelism.* Figure 4 depicts an example of pipelined and partitioned parallelism of a query execution. The work by Vertica [Vertica '09] and Teradata [Clark '00] are examples of shared-nothing parallel database management systems.

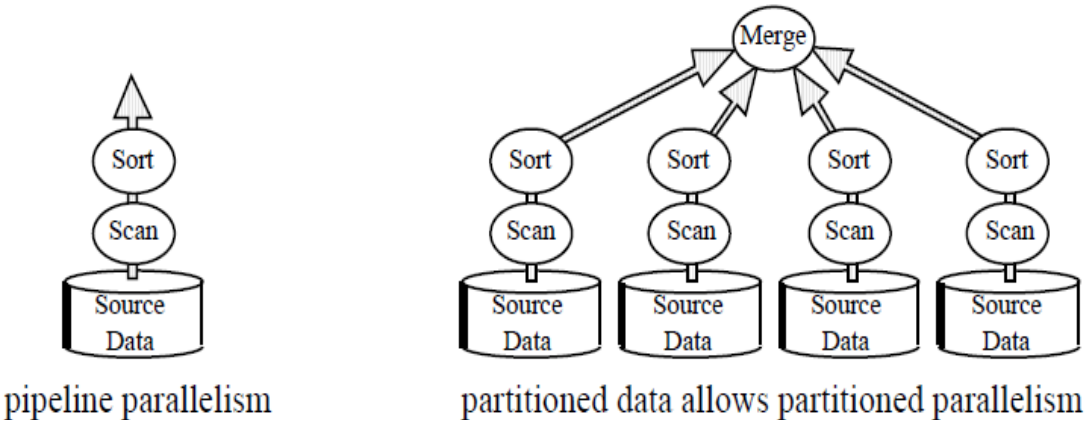

Figure 4: The dataflow approach to relational operators gives both pipelined and partitioned parallelism [Dewitt et al. '92].

4. **Stream-processing** is one of the most common ways in which graphics processing units and multi-core hosts are programmed [Gu *et al.* '09]. In the stream-processing architecture, each member of the input data array is processed independently by the same processing function using multiple computational resources. This technique is also called *Single-Program-Multiple-Data*, a term derived from Flynn's taxonomy of CPU design [Duncan '90]. Sphere [Gu *et al.* '09] is an example of a stream-processing system for data-intensive applications.

5. **Hybrid DBMS** architectures try to move relational DBMSs to the cloud. Large-scale database processing is traditionally done with shared-nothing parallel DBMSs. While shared-nothing parallel DBMSs exploit pipelined and partitioned parallelism, they suffer from several limitations including poor fault tolerance, poor scaling and a need for homogeneous platforms. There are a number of recent proposals for a hybrid approach for clouds that combines the fault tolerance, heterogeneity, and ease-of-use of MapReduce with the efficiency and performance of shared-nothing parallel DBMSs. Examples of hybrid DBMSs include Greenplum [GreenPlum], and HadoopDB [Abouzeid *et al.* '09].

3. Workload Management Taxonomy

In this section, we present a taxonomy for managing data-intensive workloads in the cloud. The taxonomy proposed provides a breakdown of techniques based on functionality. The taxonomy is used in the next section of the paper to classify and evaluate existing workload management systems. The top layer of the taxonomy, which is shown in Figure 5, contains the four main functions performed as part of workload management.

Recent systems on large-scale data processing include MapReduce [Dean *et al.* '08], Dryad [Isard *et al.* '07] and Clustera [DeWitt *et al.* '08]. Researchers have used *shared-nothing clusters*[1] for development and evaluation[2] of these systems. Other researchers such as Abouzeid *et al.* have exposed their data processing techniques to clouds; however, assuming a fixed set of resources during workload execution. We observe that elastic systems and large-scale data processing systems are disjoint at present. Since our taxonomy is based on existing literature, scheduling and provisioning, and their associated techniques and systems, are discussed separately.

[1] a collection of independent resources each with local disk and local main memory, connected together on a high-speed network.

[2] there is no reason why these approaches cannot be exposed to clouds with little effort. So, we treat these approaches as if they have been exported to clouds and only focus on the scheduling aspect of them.

Workload characterization is essential for workload management as it provides the fundamental information about a workload to the management function. Workload characterization can be described as the process of identifying characteristic classes of a workload in the context of workloads' properties such as costs, resource demands, business priorities and/or performance requirements. For example, a MapReduce workload is often characterized as a simple abstraction data processing over heterogeneous resources. MapReduce workloads are highly scalable and known to operate over thousands of resources. On the other hand, the workload of a shared-nothing parallel DBMS (PDB) is often characterized as a relational abstraction of processing structured data while operating over homogenous resources.

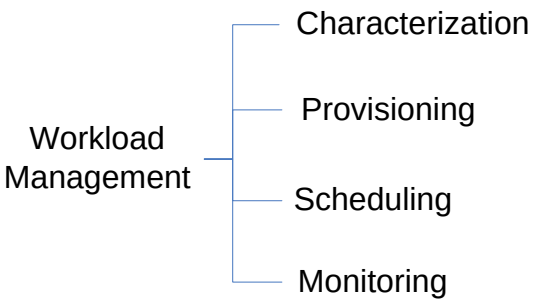

Figure 5: Taxonomy of workload management in cloud computing.

Provisioning is the process of allocating resources to workloads. The ability of clouds to dynamically allocate and remove resources implies that provisioning should be viewed as a workload management mechanism. We argue that provisioning of data-intensive workloads needs to balance workload-specific concerns such as service level objectives (SLOs) and cost with system-wide concerns such as load balancing, data placement and resource utilization.

Scheduling controls the order of executing the individual requests in workloads according to some specified objectives. In general, researchers in recent literature on large-scale data processing has assumed that the amount of resources is invariant. So, schedulers multiplex the work-units amongst the available resources. The scheduling of data-intensive workloads is impacted by the presence of multiple replicas of required data sets placed at different geographical locations, which makes it different from scheduling compute-intensive workloads.

Monitoring is essential to provide feedback to the scheduling and the provisioning processes. It tracks the performance of cloud components and makes the data available to the other processes. Monitoring can either be integrated into these processes or exist as a separate autonomous process.

In case of a separate process, monitoring requires (a) the publication of static data (e.g. number of resources) and dynamic data (e.g. current load on resources under use), (b) a global view of this data, and (c) a query mechanism capable for accessing this data. Monitoring is responsible for tracking the performance of cloud components at all times, tracking and recording throughput, response time and availability of these components from a variety of locations. It also needs to be scalable to allow hundreds of resources to publish and be resilient if any resource fails. For example, CloudWatch [Amazon 'b] provides a monitoring service. CloudWatch provides users with visibility into resource utilization, operational utilization, and overall demand patterns —including metrics such as CPU utilization, disk reads and writes, and network traffic. It is used by AutoScaling [Amazon 'c] to provision or relinquish resources.

In case of an integrated component, monitoring has a *local view*. That is, the role of the integrated monitoring is limited and customized to the needs of the scheduling. Take the example of Dryad [Isard *et al.* '07]. A simple daemon runs on each node managed by Dryad. The daemon acts as a proxy so that the job manager can communicate with the remote resources and monitor the state of the computation. A web-based interface shows regularly-updated summary statistics of a running job and can be used to monitor large computations. The statistics include the progress made, the amount of data transferred, and the error codes reported by failures. Links are provided from the summary page that allows a developer to download logs or crash dumps for further debugging. Monitoring has been explored in some detail in the research literature; however, we leave this out of this paper for brevity reasons.

The remainder of this paper focuses on the scheduling and provisioning functions of workload management in the cloud. The categories used are explained within each of the functions and then used to classify existing systems and mechanisms from the research literature. The workload characterization and monitoring functions are left for future work.

3.1. Scheduling

The scheduling portion of our taxonomy is shown in Figure 6. We present a number of key features of a scheduling approach that can be used to differentiate among various approaches. Clouds are viewed as similar to grids in a number of ways [Foster *et al.* '08] and our scheduling portion of taxonomy builds on previous studies of scheduling in grids [Dong '09; Venugopal *et al.* '06].

3.1.1. Work-unit

Scheduling policies can be classified by the *work-unit* (the job abstraction) exposed to the scheduler for execution. The scheduling portion of the taxonomy identifies two subclasses of work-units, namely tasks and workflows. Each work-unit has its own scheduling requirements. The work-units can range from simple queries (fine-grained data-intensive tasks), to coarser levels such as workflows of tasks.

3.1.1.1. Task

A task is the atomic unit for scheduling and computation to a single resource. A data-intensive *task* consists of arbitrary computation on data where data access (read or write) is a significant portion of task execution time. The data access also affects the scheduling decision. For example, the scheduler may dispatch a task to a node with weaker computational capabilities if the required data is already local to that node. We use the term data-intensive task or task interchangeably for the rest of this paper.

3.1.1.2. Workflow

A *workflow* represents a set of tasks that must be executed in a certain order because there are computational or data dependencies among the tasks. The products of the preceding tasks may be large data sets themselves (e.g., in a simple two step workflow: the first task could be a simulation and the second task could analyze the results of the simulation). Therefore, scheduling of individual tasks in a workflow requires careful analysis of the dependencies to achieve a certain objective. For the preceding example, an objective might be to reduce the amount of data transfer.

3.1.2. Objective Functions

A scheduling algorithm tries to minimize or maximize some *objective function*. The objective function can vary depending on the requirements of the users and the architecture of a specific cloud. There are two major parties in cloud computing, namely users (resource consumers) and cloud providers (resource

providers). Users are concerned with the performance of their workloads and the total cost to run their work, while cloud providers care about the utilization of their resources and revenue. Interests are captured by objective functions. Objective functions can therefore be classified as user-centric and cloud-centric.

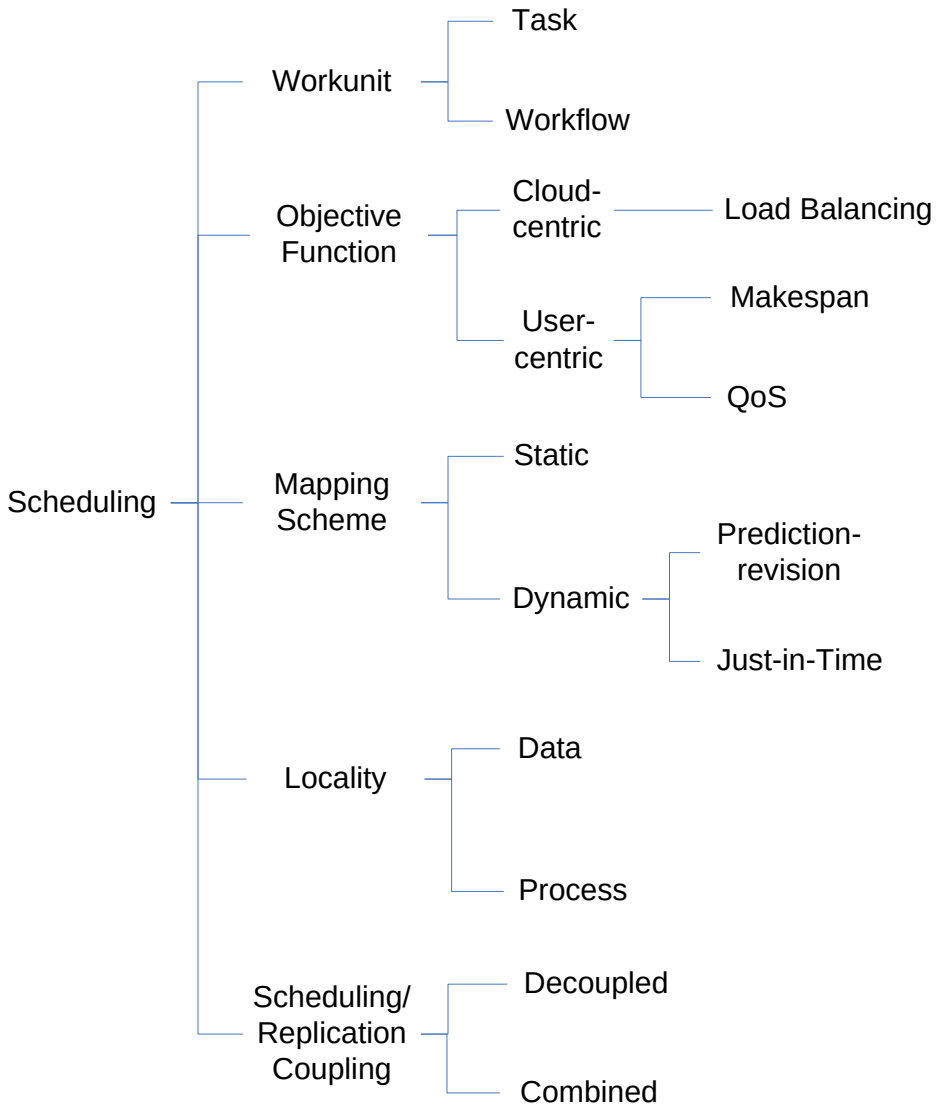

Figure 6: Scheduling portion of workload management taxonomy.

3.1.2.1. User-centric objective functions

User-centric objective functions aim to optimize the system from the user's point of view, and hence maximize performance of each work-unit. An objective function based on the *makespan* aims to minimize the average total completion time of a work-unit. Traditionally, scheduling policies are makespan based.

On the other hand, QoS[1] aims to meet performance requirements specified for a work-unit or workload such as minimum cost, an average response time or a completion deadline. Therefore, additional mechanisms may be needed, such as negotiation between users and cloud providers, service level agreements and resource reservation.

3.1.2.2. Cloud-centric objective functions

Cloud-centric objective functions aim to optimize the system from the cloud provider's point of view, and hence are concerned with maximizing revenue or resource utilization. In order to maximize revenue in a competitive cloud market, providers typically offer multiple levels of performance and reliability with different pricing. The aim of scheduling policies with a cloud-centric objective function is to provide predictable and reliable behaviour. *Load-balancing*, which distributes load in the datacenter so that maximum work can be obtained out of the physical resources, is a commonly used cloud-centric objective function.

3.1.3. Mapping Scheme

There are two basic types of methods to map work-units to resources in workload scheduling, namely static and dynamic methods. In the case of *static mapping*, the complete execution schedule is drawn prior to the actual execution of the work-unit. While in the case of the *dynamic mapping* scheme, the basic idea is to be aware of the status of execution and adapt the schedule accordingly.

3.1.3.1. Static

In the "traditional" static mode, information regarding all available resources and all work-units to be scheduled is assumed to be available before the schedule is made [Casavant *et al.* '88]. So, every work-unit is assigned once to a certain resource. Thus, the placement of a work-unit is static. In case of any unanticipated events such as failures, the execution schedule is redrawn and the work-unit is re-executed ignoring any progress made previously. Static mapping is usually based on estimates, and predictions of the behaviour – such as execution time – of work-units.

3.1.3.2. Dynamic

Dynamic mapping schemes, on the other hand, are aware of the status of execution and adapt the schedule accordingly. Dynamic mapping is usually applied when it is difficult to estimate the cost of work-units, unanticipated events are the norm, or work-units arrive on the fly. We further classify dynamic mapping schemes as prediction-revision and just-in-time schemes. *Prediction-revision* schemes create an initial execution schedule based on estimates and then dynamically revise that schedule during the execution as necessary. *Just-in-time* or *lazy* schemes do not make an initial schedule and delay scheduling decisions for a work-unit until it is to be executed [Yu *et al.* '05].

3.1.4. Locality

Exploiting the locality of data has been a key technique for scheduling and load-balancing in parallel programs [Hockauf *et al.* '98; McKinley *et al.* '96], and for query processing in databases [Shatdal *et al.* '94]. We believe it will have similar importance for scheduling of data-intensive workloads in the cloud. We

[1] QoS is a concern for many cloud applications. This concern can be seen by anticipation of cloud providers guaranteeing availability of their resources. For example, Amazon commits to 99.95% availability in a region.

identify the type of locality exploited as either data or process locality. *Data* locality involves placing a work-unit in the cloud such that the data it requires is available on or near the local host, whereas *process* locality involves placing data near the work-units. In other words, we view data locality as moving computation to data, and process locality as moving data to computation.

3.1.5. Scheduling/Replication Coupling

In a cloud environment, the location where the computation takes place may be separated from the location where the input data is stored, and the same input data may have multiple replicas at different locations. Therefore, if only the computational cost is considered in the scheduling, any gain might be offset by a high data access cost. When the interaction of scheduling and replication is considered, there are two approaches:

1. decoupling scheduling from data replication,
2. producing a combined scheduling and replication schedule.

In each case, the scheduling policy exploits locality differently.

3.1.5.1. Decoupled-Scheduling

Decoupled-scheduling manages scheduling and replication separately. That is, scheduling and replication operate independently without any direct relationship or interaction with each other. In exploiting data locality, the scheduler takes the data requirements of work-units into account and places them close to a data source for execution. Data replicas are made in the face of increased access demand on data. Data replicas may also be made for non-performance reasons such as fault-tolerance. Some systems perform "placement-aware scheduling", which is a special case of decoupled-scheduling exploiting data locality. In this case, both scheduler and replicator (if it exists) are agnostic to the need of creating any replicas in the face of increased data demands. As a possible consequence, the scheduler may overload data resources if there are no replicas [Ranganathan *et al.* '02]. In exploiting process locality, the scheduler brings input data to the work-unit and then takes away the output data or results. So, the storage requirement on the processing node is transient, that is, disk space is required only for the duration of execution.

3.1.5.2. Combined-Scheduling

Combined-scheduling manages scheduling and replication together. In exploiting data locality the scheduler creates replicas of data, either before the data is needed [Desprez *et al.* '06] or at first access [Lim *et al.* '10]. Then, the scheduler places work-units on nodes with replicas. In exploiting process locality, if the scheduler creates a replica for one work-unit then subsequent work-units requiring the same data can be scheduled on that host or in its neighborhood. However, one requirement for a resource is to have sufficient storage available to store replicas. This requirement creates a bias in the selection of compute resources. There is a possibility that resources may be disregarded only due to lack of storage space. Also, creating replicas add further overhead to work-unit execution.

3.2. Provisioning

Provisioning is the process of allocating resources for the execution of a work-unit. Clouds' support for elastic resources requires that provisioning should be viewed as a workload management mechanism. This is because resources can be dynamically allocated or de-allocated, during execution, to match the demands of a workload. This is also called dynamic provisioning. Provisioning for data-intensive workloads is further complicated by the need to move or copy data when the resource allocation changes. A user preference is the availability of a best possible QoS and transparent elasticity with no disruption of service

while minimizing costs. While a cloud provider wants to maximize revenue and the use of resources. The user and cloud goals are conflicting and workload management needs to consider tradeoffs to achieve an optimal balance. The provisioning portion of our taxonomy (Figure 7) identifies key features that can be used to categorize provisioning approaches.

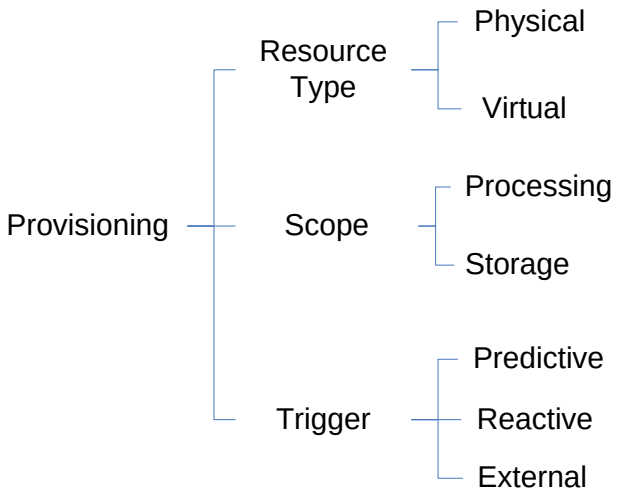

Figure 7: Provisioning portion of taxonomy

3.2.1. Resource Type

Clouds currently provision two types of resources (a) virtual and (b) physical. *Virtual resources* emulate the hardware with a virtual machine (VM). A VM is a unit of a virtual resource. A VM can be packaged together with applications, libraries, and other configuration settings into a *machine image* that gives the illusion of a particular platform while hiding the underlying hardware. In the case of *physical resources*, a user either sees the underlying hardware directly, or has knowledge of the hardware supporting the virtual resources.

3.2.2. Scope

The *scope* of provisioning is the kind of resources that are varied, that is, processing resources or storage resources. It is simple to see how the key properties of clouds, namely short-term usage, no upfront cost, and infinite capacity on demand, apply for processing resources; it is less obvious for storage resources [Armbrust *et al.* '10]. In fact, it is still an open research problem to create a highly scalable storage system with an ability to combine management of large data with the clouds to quickly scale on demand.

Processing resources are typically provisioned in terms of acquiring or relinquishing resources. For example, when an application's demand for computation increases, more VMs can be acquired and the workload multiplexed across the increased VM set. Similarly, when the demand tails off then some VMs can be released. In this manner, the workload execution performance could be maintained at a "required" level. *Storage resources* require more complicated provisioning actions such as migrating a database [Elmore *et al.* '10] or varying the number of data nodes [Lim *et al.* '10].

3.2.3. Trigger

A *trigger* is the method used to initiate provisioning. The trigger can be internal or external to the provisioning component. We identify two types of internal triggers, namely predictive and reactive triggers, which are part of the controller managing provisioning in a cloud. We also identify external triggers that can request provisioning from outside a controller. *Predictive triggers* use models to forecast a need for resource variation. These triggers anticipate the need to provision new resources and so minimize the impact of the provisioning process on workload execution. *Reactive triggers* initiate provisioning when certain conditions occur such as violation of a workload's SLOs. These conditions are set by the user. *External triggers* are driven by the decisions outside the provisioning controller for example by a user or from a scheduler. SLOs are mutual agreement on quality of service between the user and the cloud provider.

The provisioning controllers operate on a set of actuators exposed by the underlying processing or storage management system. In the case of processing resources, a controller uses cloud actuators to change the number of processing resources. In the case of storage resources, a controller does not distribute the data itself, rather it uses an existing rebalancer utility to rebalance data across the storage or data resources. This relieves the controllers from looking at data consistency issues. In this case, the focus of a controller is on the decision making on when to vary data resources, and what parameters (e.g. amount of bandwidth) should be used for rebalancing.

Predictive and reactive approaches come with their classical tradeoffs. Cheap forecasts are quick and dirty but may result in misestimation of resource variations and hence, misprovisioning in the amount of resources. Accurate forecasts need more time for calculations. In some circumstances, accurate forecasts may not be timely enough for when the variation is needed. On the other hand, the reactive method is less likely to misprovision but it may result in some disruption to workload execution while the provisioning adjusts.

4. A Survey of Workload Management Techniques and Systems

In the following sections we examine workload management systems for data-intensive workloads in a cloud present in current research literature. The taxonomy described in the previous section is used to categorize and evaluate this work. The scheduling and provisioning aspects of the systems are explored in some detail.

4.1 Scheduling

Scheduling has been a well researched topic [JSSPP]. In this paper, we focus on strategies that explicitly deal with data during processing. In presenting our survey of scheduling, we organize large-scale data processing systems according to the five data-intensive computing architectures discussed in the background section (2.3) of this paper.

4.1.1 MapReduce

Google has a MapReduce implementation [Dean *et al.* '08] called *GoogleMR*. In GoogleMR, map and reduce functions are encapsulated as tasks that perform some computation on data sets. Tasks are grouped into a workflow (MR-workflow) in which map tasks are executed to produce intermediate data sets for reduce tasks. Data is managed by the Google File System (GFS) [Sanjay *et al.* '03]. The GFS uses replication to provide resiliency against failures of machines containing data. These replicas are also exploited by GoogleMR to provide decoupled-scheduling.

The execution overview of GoogleMR is given in Figure 8. The scheduler resides on the master host. It exploits data locality by taking the location information of the input files into account, and schedules map tasks on or near a worker host that contains a replica of its input data. The map tasks process input data to

produce intermediate results and store them on the local disks. The scheduler notifies the reduce tasks about the location of these intermediate results. The reduce tasks, then, use remote procedure calls to read the intermediate results from the local disks of the map workers. The reduce tasks process the intermediate data and append the results to the final output for this data partition.

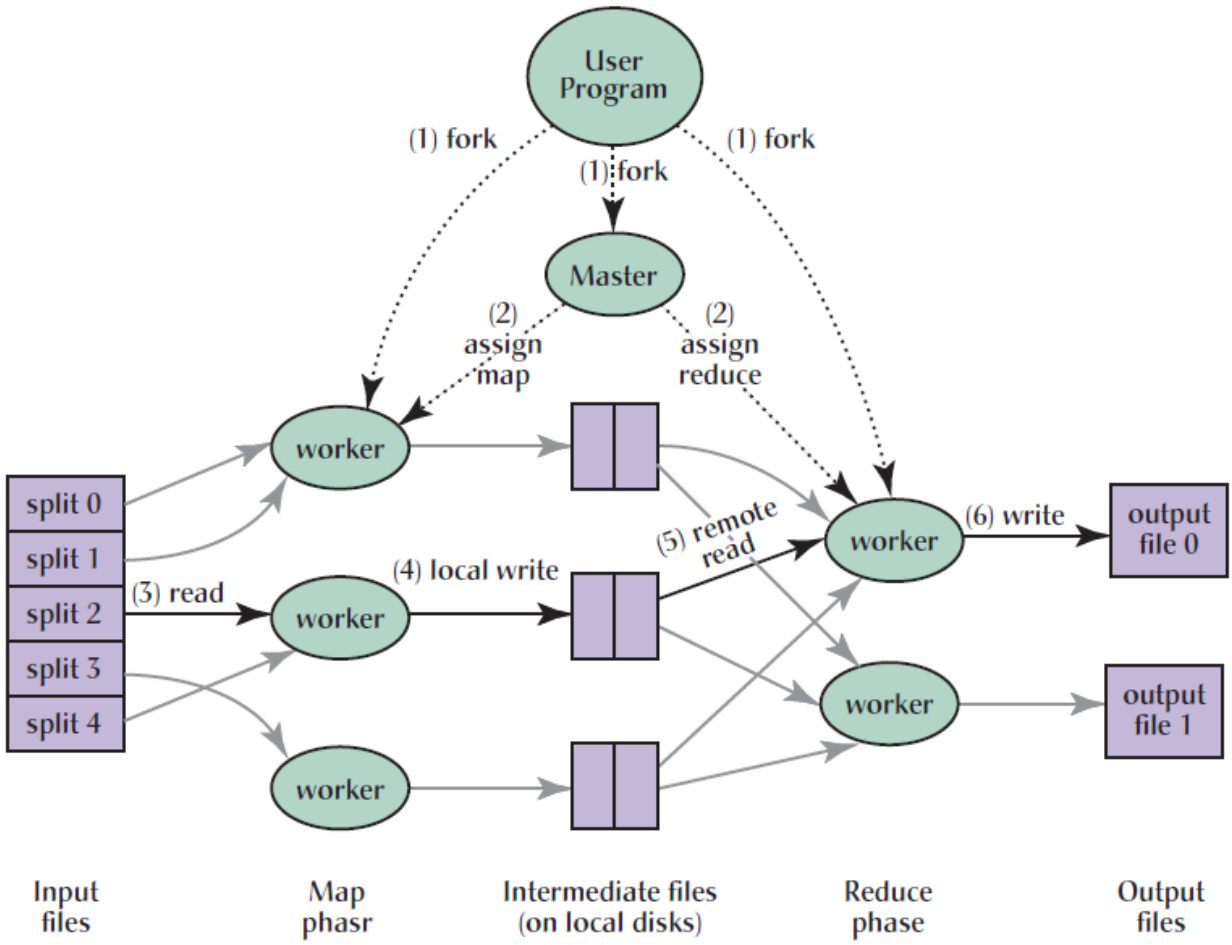

Figure 8: GoogleMR Execution Overview [Dean et al. '04].

MR-workflow execution may suffer from *execution skew*, that is, all the computation occurs in a small subset of tasks resulting in the execution time of some tasks being much greater than the others. Further, considering that the size of data is large and MapReduce is intended to be scaled to hundreds, possibly thousands of commodity PCs, failures are expected. The average task death per MR-workflow is reported as 1.2 tasks [Dean *et al.* '08]. The scheduler uses a dynamic mapping scheme to address execution skew and failures, and is likely to be just-in-time mapping. The objective function of the scheduler is to reduce the makespan of the MR-workflow.

Hadoop [Apache 'a] is an open source implementation of MapReduce that closely follows the GoogleMR model. Hadoop consists of two layers [Abouzeid *et al.* '09]: (i) a data storage layer or the Hadoop Distributed File System (HDFS) [Apache 'c], and (ii) a data processing layer based on MapReduce Framework. The input data is managed by HDFS. HDFS is a block-structured file system managed by a central NameNode. Individual files are broken into blocks of a fixed size and distributed across multiple DataNodes in the cluster. The NameNode maintains metadata about the size and the location of blocks and their replicas.

The Hadoop framework follows a simple master-slave architecture. A master is a single JobTracker and the slaves or worker nodes are TaskTrackers. A JobTracker handles the runtime scheduling of a MR-workflow and maintains information on each TaskTracker's load and available data hosts. Each MR-workflow is broken down into map tasks based on the number of data blocks that require processing, and reduce tasks. The JobTracker assigns tasks to TaskTrackers with the aim of load balancing. It achieves data locality by matching a TaskTracker to map tasks that process data local to TaskTracker.

TaskTrackers regularly update the JobTracker with their status through heartbeat messages. Hadoop's built-in scheduler runs tasks in a first-in-first-out (FIFO) order, with five priority levels [Zaharia *et al.* '09]. When a task slot becomes free, the scheduler scans through MR-workflows in order of priority and submit time to find a task of the required type. In this way, Hadoop has just-in-time mapping.

4.1.2 Dataflow-processing

Condor is a high-throughput distributed batch computing system [Thain *et al.* '05]. Condor, like other batch systems, provides a task management mechanism, scheduling policy, priority scheme, and resource monitoring and management. The *Directed Acyclic Graph Manager (DAGMan)* is a service, built on top of Condor, that is responsible for executing multiple tasks with dependencies. The coordination among data components and tasks can be achieved at a higher level by using the DAGMan. In the same manner that DAGMan can dispatch tasks to a Condor agent (or a daemon present on a computational resource), it can also dispatch data placement requests. In this way, an entire DAG or workflow can be constructed that stages data to a remote site, runs a series of tasks and retrieves the output. So, the DAGMan/Condor combination employs combined-scheduling. Since both tasks and data are dispatched, DAGMan/Condor does not subscribe to any particular locality camp and hence uses a *hybrid* approach. DAGman is an external service and only ensures that tasks are executed in the right order. Therefore, only tasks are exposed to the Condor scheduler and the mapping of tasks to hosts is performed at execution time. DAGMan also does not make an execution schedule based on estimates, so, the mapping scheme is dynamic and employs just-in-time mapping. With a workflow of tasks, a user is probably interested in reducing the makespan of the workflow rather than throughput of individual tasks in the workflow.

Dryad is a general-purpose framework for developing and executing coarse-grain data applications [Isard *et al.* '07]. It draws its functionalities from cluster management systems like Condor, MapReduce implementations, and parallel database management systems. Dryad applications consist of a dataflow graph (which is a workflow of tasks) where each vertex is a program or a task and edges represent data channels. The overall structure of a Dryad workflow is determined by its communication or dataflow. It is a logical computation graph that is automatically mapped onto data hosts by the runtime engine providing dynamic mapping. The structure of the Dryad system is shown in Figure 9. A simple daemon (D) runs on each host managed by Dryad. The daemon acts as a proxy so that the job manager (JM) can communicate with the remote resources. The job manager consults the name server (NS) to discover the list of available hosts. It maintains the dataflow graph and schedules vertices (V) on available hosts using daemons. The vertices can exchange data through files, TCP pipes, or shared-memory channels. The vertices that are currently executing are indicated by the shaded bar in the figure.

Dryad uses a distributed storage system, similar to GFS, in which large files are broken into small pieces, and are replicated and distributed across the local disks of the cluster computers. Because input data can be replicated on multiple computers in a cluster, the computer on which a graph vertex or a task is scheduled is in general non-deterministic. Moreover the amount of data written during the intermediate computation stages is typically not known before a computation begins. Therefore in such situations, dynamic refinement is often more efficient than attempting a static schedule in advance. A decoupled-scheduling approach is used that exploits data locality, and makespan is the objective function. Dryad is not a database

engine and it does not include a query planner or optimizer, therefore, the dynamic mapping scheme is just-in-time.

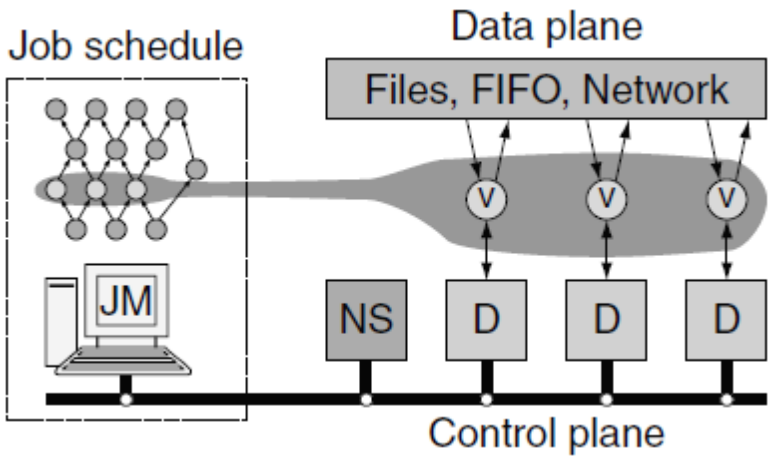

Figure 9: The Dryad system organization [Isard et al. '07].

Clustera [DeWitt *et al.* '08] shares many of the same goals as Dryad and is similarly categorized with our taxonomy. Both are targeted towards handling a wide range of work-units from fine-grained data-intensive tasks (SQL queries) to workflows of coarse-grained tasks. The two systems, however, employ radically different implementation methods. Dryad uses techniques similar to those first pioneered by the Condor project. In the Condor system, the daemon processes run on each host in the cluster to which the scheduler pushes tasks for execution. In contrast, Clustera employs a pull model. In the pull model, a data host runs a web-service client that requests work from the server. The web-service is forked and monitored by a daemon process. In addition to dataflow-processing, both Dryad and Clustera support execution of MR-workflows.

Cosmos, developed by Microsoft, is a distributed computing platform for storing and analyzing massive data sets [Chaiken *et al.* '08]. Cosmos is designed to run on large clusters consisting of thousands of commodity servers. The Cosmos Storage System, similar to GFS, supports data distribution and replication. It is optimized for large sequential I/O and all writes are append-only. Data is compressed to save storage and increase I/O throughput. Data is also distributed and replicated to provide resiliency against failure of machines containing data. Cosmos application is modeled as a dataflow graph.

SCOPE is a declarative language that allows users to focus on the data transformations that are required to solve the problem at hand while hiding the complexity of the underlying platform. The SCOPE compiler and optimizer are responsible for generating an efficient execution schedule or an optimized workflow. The optimized workflow is an input[1] for Cosmos for execution. The runtime component of the Cosmos execution engine is called the job manager. The job manager is the central and coordinating process for all tasks within a workflow. The primary function of the job manager is to map a compile time workflow to the runtime workflow and execute it.

Cosmos schedules tasks for execution on servers hosting their input data. The job manager schedules a task onto the hosts when all the inputs are ready, monitors progress, and re-executes parts of the workflow on failure. SCOPE/Cosmos is evaluated in terms of scalability of elapsed times. So, it seems that SCOPE/Cosmos aims for makespan utility. The Cosmos extensions provide the same functionality as the

[1] In contrast, only tasks were exposed to the Condor scheduler by DAGMan.

Google MapReduce. SCOPE/Cosmos provide a prediction-revision mapping and employ decoupled-scheduling approach.

4.1.3 Shared-nothing relational processing

Shared-nothing parallel database management systems (PDBs) provide relational processing in a shared-nothing setup. In a shared-nothing setup, each processor has a private memory and one or more disks, and processors communicate via a high-speed network. The structure of shared-noting design is shown in Figure 10.

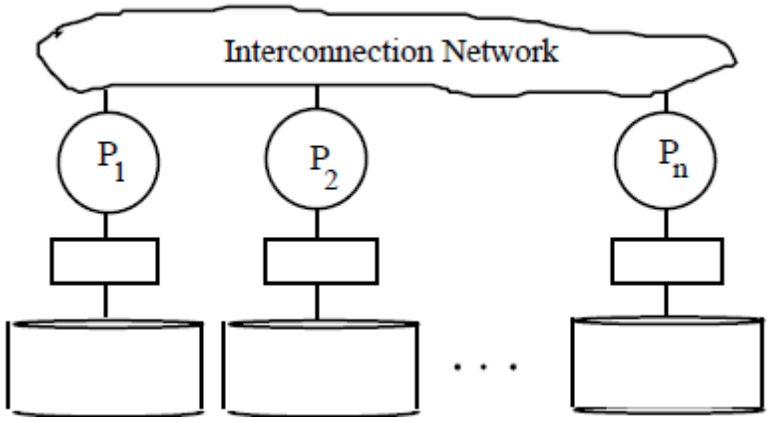

Figure 10: The basic shared-nothing design [Dewitt et al. '92].

Madden *et al.* (07) observe that shared-nothing does not typically have nearly as severe bus or resource contention as shared-memory or shared-disk machines. Therefore, they claim in a blog posting that "shared nothing [parallel DBMSs] can be made to scale to hundreds or even thousands of machines". In reality, this is highly unlikely. Abouzeid *et al.* (09) present a number of reasons to counter this claim. A prominent reason is that there is a strong trend towards increasing the number of nodes that participate in query execution to increase parallelization using cheap low end resources. However, PDBs historically assume that failures are rare events and "large" clusters mean dozens of data resources (instead of hundreds or thousands). The execution schedule is drawn prior to execution (static mapping), and any failure results in the entire relational query being re-executed. Secondly, PDBs generally assume a homogeneous set of machines. However, it is nearly impossible to get homogeneous performance across hundreds or thousands of compute nodes, even if each node runs on identical hardware or on an identical virtual machine. Finally, despite executing on homogenous resources, PDBs are prone to execution and *data skew*[1].

Abouzeid *et al.* (09) state that PDBs implement many of the performance enhancing techniques developed by the research community over the decades. Hence, PDBs best meet the "performance property" in terms of minimizing execution time of queries. In a PDB, most data is partitioned over multiple data resources in a shared-nothing cluster. However, the partitioning mechanism is transparent to the end-user. PDBs use a scheduler that turns SQL commands into an execution schedule. The scheduler optimises workload distribution in such a way that execution is divided equally among multiple homogenous data resources hosting different partitions. Data partitioning is key to the PDB performance [Dewitt *et al.* '92]. There are various data partitioning schemes; most prominent are round-robin, range and hash partitioning. Certain data access patterns may influence the choice of a particular data partitioning scheme. Nonetheless, these schemes operate independently of scheduling of tasks to data hosts conforming to decoupled-scheduling in our taxonomy.

[1] Data skew is when all the data is present in one or few partitions rather than evenly distributed across all the data resources.

4.1.4 Stream-processing

Sector [Gu *et al.* '09] is a distributed storage system that operates over high-speed network connections. Sector has been deployed over a wide geographical area and it allows users to obtain large data sets from any location. For example, Sector has been used to distribute an astronomical data set (Sloan Digital Sky Survey [SDSS]) to astronomers around the world. In addition to the ability of operating over wide areas, Sector automatically replicates files for better reliability, availability and access throughout the WAN.

Sphere [Gu *et al.* '09] is a compute service built on top of Sector and provides a set of simple programming interfaces for users to write distributed data-intensive applications using a stream abstraction. A Sphere stream consists of multiple data segments and which are processed by Sphere Processing Engines (SPEs) using hosts. A SPE can process a single data record from a segment, a group of data records or the complete segment. User-defined functions (UDFs) are supported by the Sphere cloud over data both within and across datacenters.

Data-intensive applications could be executed in parallel in two ways. First, the Sector data set which consists of one or more physical files can be processed in parallel. Second, Sector is typically configured to create replicas of files for archival purposes. These replicas can also be processed in parallel. An important advantage provided by Sphere is that data can often be processed in place without moving it, and hence achieving data locality.

The computing paradigm of Sphere is shown in Figure 11. The SPE is the major Sphere service or a task and is started by a Sphere server in response to a request from a Sphere client or user. Each SPE is based on a user-defined function. Each SPE takes a segment from a stream as input and produces a segment of a stream as output. These output segments can themselves be the input segments to other SPEs.

Contrary to the different systems discussed so far, a user is responsible for orchestrating the complete running of each task in Sphere. One of the design principles of the Sector/Sphere system is to leave most of the decision making to the user, so that the Sector master can be quite simple. The objective function of Sector/Sphere system is therefore user-centric, and makespan is used as an example by Gu *et al.* (09). In Sphere, the user is responsible for the control and scheduling of the program execution, while Sector independently replicates for parallelism.

SPEs periodically report the progress of the processing to the user. If an SPE does not report any progress before a timeout occurs, then the user abandons the SPE. The segment being handled by the abandoned SPE is assigned to another SPE if available, or placed back into the pool of unassigned segments. This way Sphere achieves fault tolerance. Sphere does not check point SPE progress; when the processing of a data segment fails, it is completely reprocessed by another SPE.

Usually the number of data segments is much larger than the number of SPEs. As a result, the system is naturally load balanced because the scheduler keeps all SPEs busy most of the time. Imbalances in system load occur only towards the end of the computation when there are fewer data segments to process, resulting in some SPEs becoming idle. Each idle SPE is assigned to one of the incomplete segments. A user collects results from the SPE that finishes first. This approach is similar to stragglers for GoogleMR [Dean *et al.* '08]. It implies that Sphere employs just-in-time mapping. There are several reasons for SPEs having different duration to process data segments including (a): the hosts may not be dedicated, (b) the hosts may have different hardware configurations (Sector hosts can be heterogeneous), and (c) different data segments may require different processing times. Gu *et al.* (09) argue that both stream-processing and MapReduce are ways to simplify parallel programming and that MapReduce-style programming can be implemented in Sphere by using a map UDF followed by a reduce UDF.

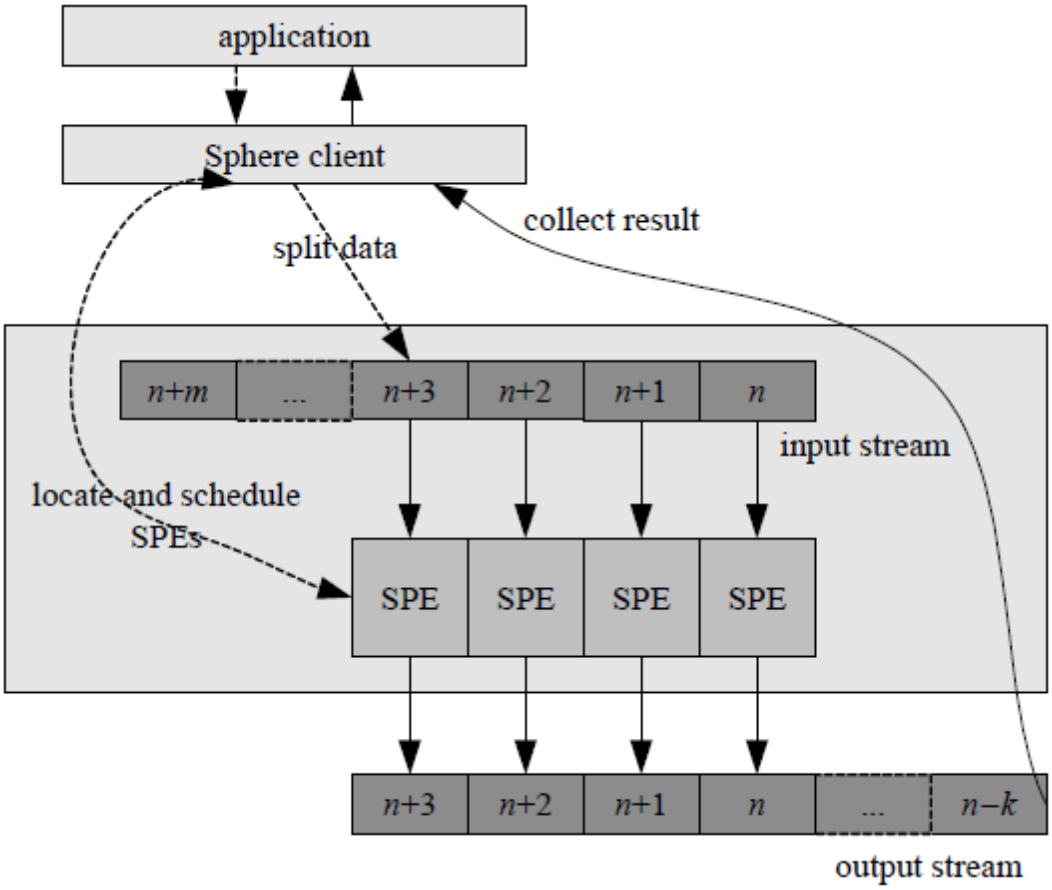

Figure 11: The computing paradigm of Sphere [Gu et al. '09].

4.1.5 MR+DB Hybrid

Driven by the competition in the open market and embracing decades of research work, PDBs best meet the "performance property" [Abouzeid *et al.* '09]. However, there are problems. First, most PDBs re-execute a failed query due to their operating environment. In their environment, queries take no more than a few hours and run on no more than dozens of resources. Query failures are relatively rare in such an environment, so an occasional query restart is acceptable [Abadi '09]. Second, even though the operating environment consists of homogeneous equipment, PDBs are prone to execution and data skew. There is a possibility of significantly poor performance if a small subset of nodes in the cluster is performing particularly poorly.

MapReduce provides high fault tolerance and the ability to operate in a heterogeneous environment [Abouzeid *et al.* '09]. In MapReduce, the fault tolerance is achieved by detecting and reassigning failed tasks to other available resources in the cluster, while the ability to operate in a heterogeneous environment is achieved via redundant task execution. In this way, the time to complete a task becomes equal to the time for the fastest resource to complete the redundantly executed task. By breaking tasks into small, granular tasks, the effect of faults and straggler resources can be minimized. MapReduce has its share of shortcomings [Abadi '09]. Long start-up time to get to peak performance, which is four to six times slower than the read rate of fast disks in the cluster; there indeed is room for improvement. Pavlo *et al.* (09) have also empirically shown that MapReduce is relatively slower than alternative systems.

More recently researchers have looked into bringing ideas together from MapReduce and database systems. However, such work focuses mainly on language and interface issues. The Hadoop's *DBInputFormat* [Apache 'b] allows users to easily use relational data as input for their MR-workflow [Gruska *et al.* '10]. The *Pig* project at Yahoo [Olston *et al.* '08] and the open source *Hive* project [Thusoo *et al.* '09] integrate declarative query constructs from the database community into MapReduce software to allow greater data independence, code reusability, and automatic query optimization. The DBInputFormat, Pig and Hive use Hadoop as the underlying MapReduce framework.

Recent Hadoop releases added a DBInputFormat component to its distribution. DBInputFormat is a class provided by Hadoop to read data from a relational database. The DBInputFormat could be used as an input format for MR-workflow. DBInputFormat is built into Hadoop and is classified the same way as Hadoop.

A Pig Latin program is compiled by the Pig system into a sequence of MapReduce tasks that are executed using Hadoop [Olston *et al.* '08]. Unlike DBInputFormat, Pig Latin is an additional abstraction layer built on top of Hadoop. Olston *et al.* (08) describe Pig Latin as a dataflow language using a nested data model. Each step in a model specifies a single, high-level data transformation. Pig Latin is classified the same way as Hadoop.

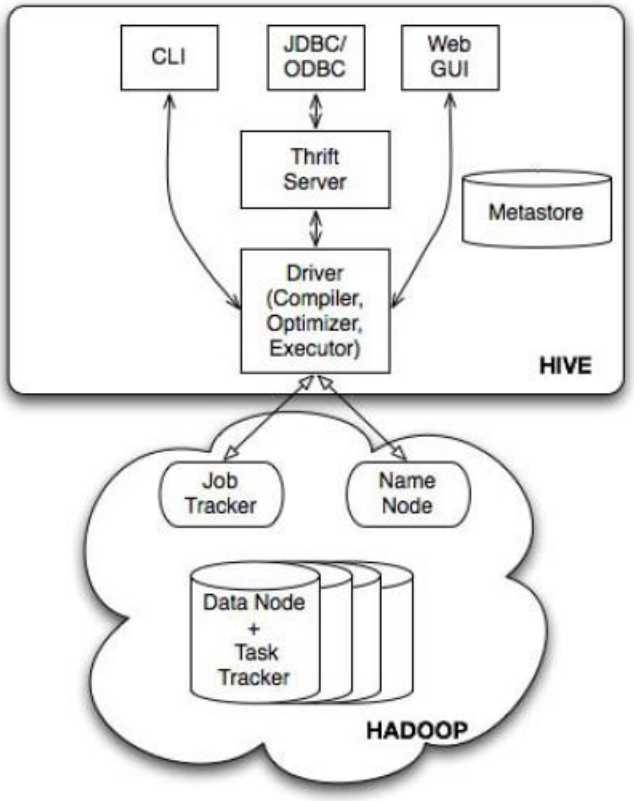

Figure 12: Hive Architecture [Thusoo *et al.* '09].

Hive is an open-source data warehousing solution built on top of Hadoop [Thusoo *et al.* '09]. Hive supports queries expressed in a SQL-like declarative language – *HiveQL*. The queries are compiled into MapReduce

tasks and are executed on Hadoop. The architecture of Hive is shown in Figure 12[1]. Hive exposes two kinds of interfaces (a) user interfaces like command line (CLI) and web GUI, and (b) application programming interfaces (API) like JDBC and ODBC. The Hive Thrift Server exposes a very simple client API to execute HiveQL statements. In addition, HiveQL supports custom MapReduce scripts to be plugged into queries. The Driver manages the life cycle of a HiveQL statement during compilation, optimization and execution. Hive also includes a system catalogue, Hive-Metastore, containing schemas and statistics, which is useful in data exploration and query optimization. Like Pig Latin, Hive is an additional abstraction layer built on top of Hadoop and is classified the same way as Hadoop in our survey.

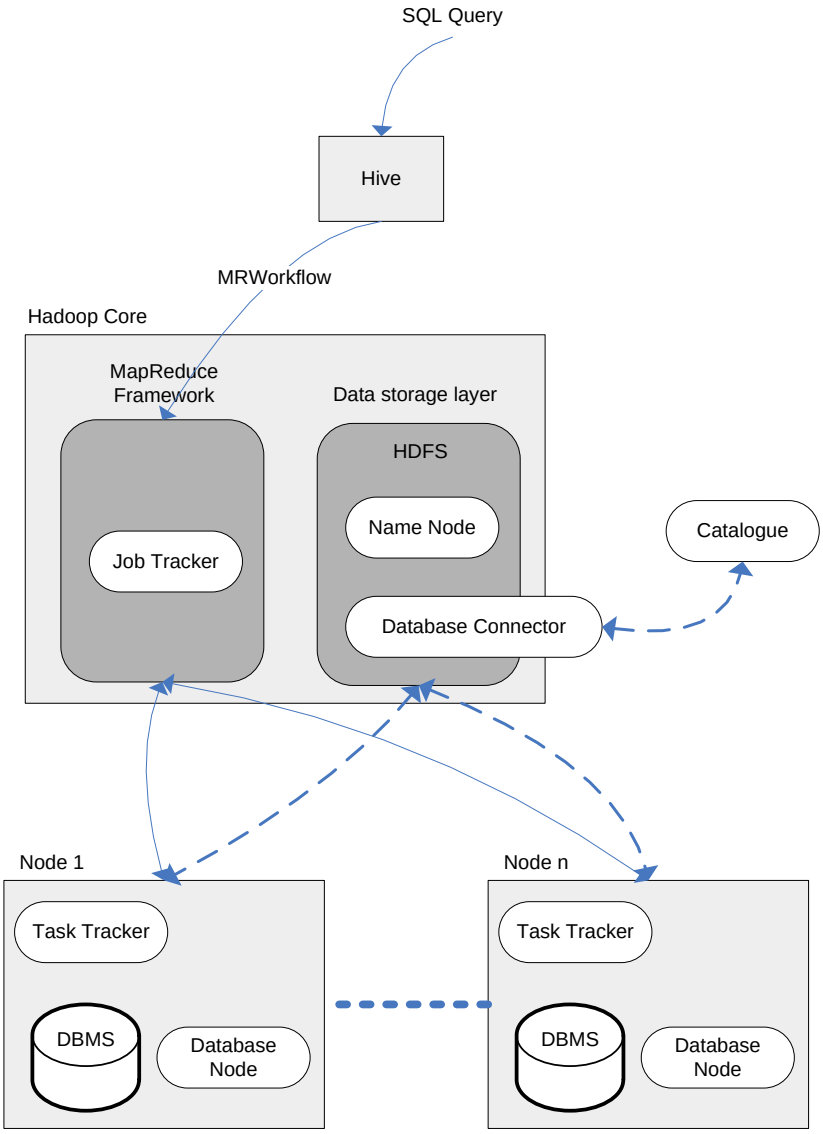

Figure 13: The Architecture of HadoopDB

[1] The Hive architecture is typical of bringing together ideas from MapReduce and database systems at the language and interface level.

Greenplum and Aster Data are analytical data management systems that offer the ability to write MapReduce functions in addition to SQL over data stored in their PDB products [Abouzeid *et al.* '09]. All of the projects discussed in MR+DB hybrid category so far are an important step in the hybrid direction at the interface level. Nevertheless, there remains a need for a hybrid solution at the systems level. *HadoopDB* [Abouzeid *et al.* '09] is one example that provides such a hybrid structure at the systems-level. The architecture of HadoopDB is depicted in Figure 13. It uses MapReduce as the communication layer above multiple data nodes running single-node DBMSs. Queries expressed in SQL are translated into MapReduce (Hadoop) tasks by extending existing tools (Hive), and then pushed into the higher performing single-node DBMSs. HadoopDB connects multiple single-node DBMSs systems using Hadoop as the task coordinator and network communication layer. It inherits scheduling and task tracking from Hadoop, while doing much of the query processing inside of the DBMS. So, it inherits fault tolerance and the ability to operate in heterogeneous environments from MapReduce and the performance offered by parallel execution from DBMSs. Data is loaded from HDFS into data node databases according to metadata exposed in the Catalog component of HadoopDB. Scheduling from Hadoop and data in data node databases are managed separately conforming to decoupled scheduling. Tasks are dispatched to data node DBMSs containing the data to exploit data locality. For the purpose of our report, HadoopDB is classified the same way as Hadoop.

4.2 Discussion

A summary of the scheduling architectures discussed in this paper is given in Table 2. We know that moving large volumes of data is expensive and causes significant delays in its processing. Moving terabytes of data over a WAN with current network technologies would require days [Armbrust *et al.* '10]. Any middleware to enable execution of data-intensive application is bottlenecked by network capacity no matter how efficient. As a result, almost all of the surveyed systems in this paper exploit data locality by bringing computations to the data source or near it. The issues related to data movement over a network suggest that bringing computation to data is a better and more appropriate approach for data-intensive workload management.

We see many schedulers employ decoupled-scheduling approach. That is, the schedulers operate independently of replication and place tasks close to data, ideally on the same resource hosting data or near it. Some schedulers specifically employ placement-aware scheduling. As described earlier, the placement-aware scheduling is a special case of decoupled-scheduling where the scheduler and replicator are agnostic to the need of creating any replicas in the face of increased data demand. In such a situation, a placement-aware scheduler may overload data resources if there are no replicas [Ranganathan *et al.* '02]. Thus, creating replicas for performance reasons is a good idea. However, there is a need to explore different replication strategies.

Using low cost unreliable commodity hardware to build shared-nothing clusters has its benefits. However, the probability of a node failure during data processing increases rapidly. This problem gets worse at larger scales: the larger the amount of data that needs to be processed, the more resources are required to participate. Further, if the resources deployed are low cost and unreliable the chances of system failures are amplified. Therefore, fault-resiliency must be built into schedulers to addresses such issues.

Most of the systems surveyed in this paper use workflow as a unit of execution and employ just-in-time mapping. This mapping approach is scalable and adapts to resource heterogeneity and failures. Nevertheless, we believe that a system could benefit from prediction-revision mapping techniques that incorporate some pre-execution planning, workflow optimization, heuristics or history analysis. This additional analysis could help in creating an appropriate number of replicas or determining an appropriate amount of resources required for a computation.

Makespan is the prevalent objective function in the survey. Clouds, however, are competitive and dynamic market systems in which users and providers have their own objectives. We therefore believe that objective functions related to cost and revenue, or participants' utilities, are appropriate and require further study. Because the economic cost and revenue are considered by cloud users and cloud providers, respectively, objective functions and scheduling policies based on them need to be developed.

Most systems surveyed here use shared-nothing clusters for large-scale data processing. We argue that these systems can be exposed to clouds with little effort. These systems assume a static resource base, whereas clouds are elastic. So, there are two immediate research opportunities in this direction. First, expose these systems on clouds. Second, make them aware of elasticity during execution. HadoopDB uses Amazon's EC2 but does not use elasticity during execution of the MR-workflow. Similarly, Amazon has made Hadoop available in its cloud with Elastic MapReduce [Amazon 'e], however the number of VMs have to be selected before the execution starts.

A summary of the evaluation using our scheduling taxonomy is given in Table 1.

Architecture	**System**	**objective function**	**mapping**	**Scheduling/ Replication Coupling**	**locality**	**work-unit**
MapReduce	GoogleMR [Dean *et al.* '08]	user->makespan; cloud-> load balancing	just-in-time	decoupled	data	workflow
	Hadoop [Apache 'a]	user->makespan; cloud-> load balancing	just-in-time	decoupled	data	workflow
Dataflow-processing	DAGMan/Condor [Thain *et al.* '05]	user->makespan	just-in-time	combined	*hybrid*[1]	task
	Dryad [Isard *et al.* '07]	user->makespan	just-in-time	decoupled	data	workflow
	Clustera [DeWitt *et al.* '08]	user->makespan	just-in-time	decoupled	data	workflow
	SCOPE/Cosmos [Chaiken *et al.* '08]	user->makespan	prediction-revision	decoupled	data	workflow
Relational Dataflow-processing	PDB [Dewitt *et al.* '92]	user->makespan	static	decoupled	data	workflow
Stream-processing	Sector/Sphere [Gu *et al.* '09]	user->makespan; cloud-> load balancing	just-in-time	decoupled	data	task
Hybrid MR+DB	Hadoop's DBInputFormat [Apache 'b]; Pig Latin[1] [Olston *et al.* '08]; Hive [Thusoo *et al.* '09]	user->makespan; cloud-> load balancing	just-in-time	decoupled	data	workflow
	HadoopDB [Abouzeid *et al.* '09]	user->makespan; cloud-> load balancing	just-in-time	decoupled	data	workflow

Table 2: Summary of the scheduling in large-scale data processing systems

[1] Both tasks and data are dispatched so this system does not subscribe to any particular locality camp.

4.3 Provisioning Techniques

We discuss provisioning at the infrastructure level of a cloud and identify three provisioning techniques currently in use, namely scaling, migration and surge computing. Therefore, our discussion of IaaS would seem obvious. This discussion is also relevant to PaaS and SaaS, since both of these may also vary the amount of resources behind the scenes. Our presentation of provisioning mechanisms in clouds for data-intensive workloads is organized based on the provisioning technique used.

4.3.1 Scaling

Scaling is a process of increasing or decreasing the amount of resources allocated during workload execution. These resources can be processing resources for computation, or storage resources for data requirements. Currently, scaling is one of the most prevalent mechanisms for dealing with variations in the workload. Commercial clouds typically offer customers the choice of a small number of fixed configuration VM types that differ in their computational capacity [Quiroz *et al.* '09]. Given fixed configuration VM types, scaling is a more effective mechanism to deal with workload demand variation. That is, expand the resource set (scale out) when workload demand increases, and reduce the resource set (scale in) when the demand tails off.

Amazon EC2 provides scaling of virtual processing resources called instances. An EC2 *instance* is primarily a virtual processing resource (VM) in the Amazon cloud. A set of instances is monitored by a web service called CloudWatch [Amazon 'b], and automatically scaled in or out by AutoScaling [Amazon 'c] according to user-defined conditions. AutoScaling takes an action based on metrics exposed by CloudWatch. In this case, the trigger is reactive since action is taken when a condition is met.

The process of instantiating new VMs could take as long as few “minutes” [Amazon 'a]. The new VMs originate either as fresh boots or replicas of a template VM, unaware of the current application state. This forces users into employing ad hoc methods to explicitly propagate application state to new VMs [Lagar-Cavilla *et al.* '09]. The adhoc methods could either impact the parent VM and hence workload execution, while the state is being propagated, or waste resources if VMs are pre-provisioned.

The *Snowflock* project [Lagar-Cavilla *et al.* '09] introduced a *VM fork* mechanism that instantaneously clones a VM into multiple replicas that execute on different hosts. All replicas share the same initial state, matching the intuitive semantics of stateful worker creation. In doing so, the VM fork provides a straight forward creation and efficient deployment of stateful workers in a cloud environment. The stateful workers start-up rapidly (< 1 second). The state is replicated to clones as “one way” [Lagar-Cavilla *et al.* '10]. That is, the children inherit the parent’s state at the time of cloning but any changes in children’s state is not propagated back to the parent. This is because there is no write-through channels back to the parent. Also, the VM fork aims at cloning VMs providing virtual processing resources leaving any large data replication and distribution policies to the underlying storage stack. The VM fork is triggered externally by the user.

S3 is Amazon’s Simple Storage System [Amazon 'h]. Conceptually, S3 is an infinite store for objects of variable size (minimum 1 Byte, maximum 5 GB) [Brantner *et al.* '08] where data is written and read as objects. Each object is stored in a bucket and retrieved via a unique, user-assigned key. That is, the user specifies which bucket an object is stored in. So, the fundamental unit of storage is a bucket. Amazon has not published details on the implementation of S3. We believe these buckets are likely virtual units of storage that are mapped down to physical media such as hard disks.

The data access throughput varies with the number of clients of a S3 bucket. In case of increased client set, the combined access throughput of all clients increase; however, access throughput for any particular client decreases [Garfinkel '07]. The decrease seems proportional to the number of concurrent clients.

Consequently, the user has to provide replicated copies of objects to maintain a consistent throughput for any given client, creating additional buckets if necessary, and copying the data. Therefore, the trigger is external for storage increase or decrease. When the number of clients decreases, the user has to delete additional copies of data while ensuring any data consistency.

Elastic storage provides elastic control for a multi-tier application service that acquire and release resources in discrete units, such as VMs of pre-determined configuration [Lim *et al.* '10]. It focuses on elastic control of a storage tier where adding or removing a data node requires rebalancing stored data across the data nodes (which consists of VMs). The storage tier presents new challenges for elastic control, namely delays due to data rebalancing, interference with application and sensor measurements, and the need to synchronize variation in resources with data rebalancing. Many previous works vary a "continuous" resource share allotted to a single data node; clouds with per-VM pricing like EC2 do not expose this actuator. So, Lim *et al.* (10) employed an integral control technique called proportional thresholding to regulate the number of discrete data nodes in a cluster. They employed a reactive controller that decides resizing node set based on a feedback signal of CPU utilization.

Google AppEngine [Google] scales a user's applications automatically for both processing resources and storage. The scaling is completely transparent to the user. The system simply replicates the application enough times to meet the current workload demand. A reactive trigger is likely used to initiate the scaling. The amount of scaling is capped in that resource usage of a given application is monitored and is not allowed to exceed its quota. There is a base level of usage available for free whereas a payment system is available to pay for higher quotas. The monitored resources include incoming and outgoing bandwidth, CPU time, stored data, and email recipients.

Microsoft Windows Azure does not offer automatic scaling but it is the primary tool for provisioning. Users can provision how many instances they wish to have available for their application. Like Amazon, the instances are virtual processing resources [Chappell '09].

4.3.2 Migration

Migration is a workload management technique used in clouds where an application execution is moved to a more appropriate host. Clark *et al.* (05) explores one of the major benefits enjoyed by virtualization, that is the migration of live OS (i.e. an OS continues to operate during migration). Live migration of virtual machines has been shown to have performance advantages in the case of computation-intensive workloads [Voorsluys *et al.* '09] as well as fault tolerance benefits [Prodan *et al.* '09].

Clark *et al.* (05) discusses a number of reasons for why a migration at the VM level is useful. First, migrating an entire OS and all of its applications as one unit avoids many of the difficulties faced by process-level migration approaches. Second, migrating at the level of an entire VM means that in-memory state can be transferred in a consistent and efficient fashion. In practical terms this means that, for example, it is possible to migrate an on-line game server or streaming media server without requiring clients to reconnect: something not possible with approaches which use application level restart. Third, live migration of VMs allows a separation of concerns between the users and providers of a data center (or a cloud). Users have complete control regarding the software and services they run within their VMs, and need not provide the operator with any OS-level access. Similarly the providers need not be concerned with the details of what is happening inside a VM. Instead they can simply migrate the entire VM and its attendant processes as a single unit.

Xen live migration [Clark *et al.* '05] achieves impressive performance with minimal application execution downtimes, and it demonstrates the migration of entire OS instances on a commodity cluster. Xen live migration uses a precopy mechanism that iteratively copies memory to the destination [Luo *et al.* '08].

Precopy takes place while the VM is still executing applications. A record of any memory pages modified after the precopy process is kept. Then at the right time, Xen suspends the VM to copy to the destination the CPU state and the remaining memory pages that were modified after precopy. It resumes the VM at the destination after all the memory has been synchronized. Since only a few pages are transferred during VM pausing, the downtime is usually too short for a user to notice – service downtimes as low as 60 milliseconds (ms) on a Gigabit LAN [Clark *et al.* '05]. The performance of live migration is sufficient to make it a practical tool even for servers running interactive loads. Any failures during migration result in the abortion of migration and the VM continues to be operative on the source host. Nonetheless, Clark *et al.* (05) assume network attached storage (NAS) is available for live VM migration and do not cater for migrating VM local storage. Consequently, only a subset of a VM is migrated. The subset contains an entire OS and all of its applications as one unit. Since the VMs assume a shared storage, no data in local (VM specific) storage is migrated. Therefore, the scope of Xen live migration is processing. Clark *et al.* (05) claim that most modern data centers consolidate their storage requirements using a NAS device, in preference to using local disks in individual VMs. This claim predates the birth of cloud computing, and we note that this is not the case with Amazon EC2.

In Amazon EC2, VMs have their local storage. Another storage capacity is made available that could be shared through mounting called *Elastic Block Storage (EBS)* [Amazon 'i]. Amazon also offers a storage cloud (S3). It is most likely that VMs will continue to have their local storage. Also, we will quite possibly see the combination of both shared and local storage being used effectively. Therefore, migration with both local storage and shared storage is relevant.

Migration with data-intensive workloads, however, faces problems with high overhead and long delays because the large data sets may also have to be moved [Elmore *et al.* '10]. Luo *et al.* (08) consider migrating *whole-system* state of a VM from the source to the destination machine, including its CPU state, memory data, and local disk storage data. During the migration time the VM keeps running. Whole-system VM migration builds on Xen live migration. In addition to the normal activities that Xen live migration performs during the migration, a block-bitmap data structure is used to track all the write accesses to the local disk storage. The block-bitmap is used for synchronizing the local disk storage in the migration process. Experiments show this setup works well even when I/O intensive workloads are running in the migrated VM. The downtime of the migration is around 100 ms, close to Xen live migration. Like Xen live migration, whole-system migration deals with VMs. However, the VMs are migrated along with data in local storage. Therefore, the scope is hybrid.

Elmore *et al.* (10) analyze various database multi-tenancy models and relate them to the different cloud abstractions to determine the tradeoffs in supporting multi-tenancy. So, the scope is the storage tier. At one end of the spectrum is the shared hardware model which uses virtualization to multiplex multiple data nodes on the same host with strong isolation. In this case, each data node has only a single database process with the database of a single tenant. At the other end of the spectrum is the shared table model which stores multiple tenants' data in shared tables providing the finest level of granularity.

Elmore *et al.* (10) provide preliminary investigation and experimental results for various multi-tenancy models and forms of migration. For shared hardware migration, using VM abstracts the complexity of managing memory state, file migration and networking configuration. Live migration only requires Xen be configured to accept migrations from a specified host. Using Xen and a 1 Gbps network switch, Elmore *et al.* are able to migrate an Ubuntu image running MySQL with a 1 GB TPC-C database between hosts on average in only 20 seconds. The authors also observe an average increase of response times by 5-10% when the TPC-C benchmark is executed in a VM compared to no virtualization. In this case, the resource type is virtual.

On the other hand, shared table migration is extremely challenging and any potential mechanism is coupled to the implementation. Isolation constructs must be available to prevent demanding tenants from degrading system wide performance in systems without elastic migration. Some shared table models utilize tenant identifiers or entity keys as a natural partition to manage physical data placement [Chang *et al.* '08]. Lastly, using a 'single' heap storage for all tenants [Weissman *et al.* '09] makes isolating a data cell for migration extremely difficult. Without the ability to isolate a data cell leaves efficient migration of shared tables an open problem. One avenue of research is investigating the right level of abstraction for database migration. Another is autonomic management and decision making of when to migrate. The trigger is external in database multi-tenancy models.

Elmore *et al.* (10) describe different methods of migrating multi-tenancy models of a database while leaving out any discussion on ACID properties provided by these models. As a result, they seem to make an implicit assumption that a DBMS on a cloud guarantees ACID (or at least some variant) properties. As discussed in the background of this paper, a transactional database on a cloud guaranteeing ACID properties is difficult to achieve. To date there has not been a concrete and demonstratable prototype [Abadi '09].

4.3.3 Surge Computing

Surge computing is a provisioning technique applicable in hybrid (private/public) clouds [Armbrust *et al.* '09]. A private cloud model saves costs by reusing existing resources, keeping some data onsite and allowing more control on certain aspects of the application. The resources for a private cloud are augmented on-demand (in times of load spikes) with resources from the public cloud. In these scenarios more than one cloud are typically connected by a WAN resulting in latency implications with moving data to the public cloud.

Zhang et al. (09) present a comprehensive workload management framework for web based applications called *Resilient Workload Manager (ROM)*. ROM includes components for (a) load balancing and dispatching, (b) offline capacity planning for resources, and (c) enforcing desired QoS (e.g. response time). It features a fast workload classification algorithm for classifying incoming workload between a base workload (executing on a private cloud) and trespassing workload (executing on a public cloud)[1]. This implies that the scope of ROM is to vary processing resources. Resource planning and sophisticated request-dispatching schemes for efficient resource utilization are only performed for the base workload. The private cloud runs a small number of dedicated hosts for the base workload, while VMs in the public cloud are used for servicing the trespassing workload. So, the resource type is hybrid. The data storage in the private cloud is decoupled from that in the public cloud so shared or replicated data is not needed.

In the ROM architecture, there are two separate load balancers one for each type of workload. The base load balancer makes predictions on the base workload and uses integrated offline planning and online dispatching schemes to deliver the guaranteed Quality of Service (QoS). The prediction may also trigger an overflow alarm. In case of an alarm, workload classification algorithm sends some workload to the public cloud for processing. ROM operates an integrated controller and load balancer in the public cloud. The controller reacts to the external alarm and provisions VMs and the load balancer services trespassing workload on the provisioned images using round-robin policy.

[1] *Base* workload refers to the smaller and smoother workload experienced by the application platform "most" of the time (e.g., 95% of the time), while *trespassing* workload refers to the "short" and transient workload spikes experienced at rare times (e.g., the 5% of the time).

Moreno-Vozmediano *et al.* (09) analyze the deployment of generic clustered services on top of a virtualized infrastructure layer that combines the *OpenNebula* VM manager [Sotomayor *et al.* '09] and Amazon EC2. The separation of resource provisioning, managed by OpenNebula, from workload management, provides elastic cluster capacity. The capacity is varied by deploying (or shutting down) VMs on demand, either in local hosts or in remote EC2 instances. The variation in the number of VMs in OpenNebula is requested by an external provisioning module. For example, a provisioning policy limits the number of VMs per host to a given threshold. The variation in the number of VMs in OpenNebula is requested by an external provisioning module. For example, a provisioning policy limits the number of VMs per host to a given threshold.

Two experiments, operating over the hybrid cloud, are reported by Moreno-Vozmediano *et al.* (09). One shares insights in executing a typical high throughput computing application, and the other at latencies in a clustered web server. For the experiments, the scope is variation in processing resources.

Technique	**System**	**Scope**	**Trigger**	**Resource Type**
Scaling	Amazon AutoScaling [Amazon 'c]	processing	reactive	virtual
	SnowFlock [Lagar-Cavilla *et al.* '09]	processing	external	virtual
	Amazon S3 [Amazon 'h]	storage	external	virtual
	Elastic Storage [Lim *et al.* '10]	storage	reactive	virtual
	Google AppEngine [Google]	hybrid	reactive	physical
	Microsoft Windows Azure [Chappell '09]	processing	external	virtual
Migration	Xen Live Migration [Clark *et al.* '05]	processing	external	virtual
	Whole-system VM migration [Luo *et al.* '08]	hybrid	external	virtual
	Multi-tenant DB Migration [Elmore *et al.* '10]	storage	external	virtual
Surge Computing	Resilient Workload Manager (ROM) [Zhang *et al.* '09]	processing	external	hybrid
	OpenNebula [Sotomayor *et al.* '09]	processing	external	virtual

Table 3: Summary of provisioning for large-scale data processing

4.4 Discussion

A summary of our classification of provisioning techniques is given in Table 3. We observe that most of the current work that is related to provisioning in clouds involves scaling. Such work is applied to web applications that do not require large-scale data processing. In the ROM system, the data storage in a private cloud is decoupled from that in the public cloud so that the latter is not tied to the former through shared or replicated data resources. This seems to be a reasonable approach for larger and read-only data.

Maintaining data consistency for read/write operations between sites in a hybrid cloud is still an open problem.

Note that some of the reactive techniques, examined here, involve a user defining rules in terms of condition and action pairs to control the reaction. With multiple rules, many questions arise such as can multiple rules be defined on the same metrics, can they overlap or contradict.

We looked at some migration approaches. Migration approaches may benefit users and cloud providers in different ways. For example, from the user's perspective, a VM may be placed in a more suitable environment such as on a resource hosting the data needed by the application residing in the VM. From the cloud provider's perspective, VMs may be rearranged across machines in a datacenter to relieve load on congested hosts for example. In such situations the combination of virtualization and migration could significantly improve manageability of hosts. Also, migration is a powerful tool for datacenter administrators, allowing separation of hardware and software considerations. If some hardware needs to be removed from service, an administrator could migrate the VMs including the applications that they are running to an alternative host, freeing the original host for maintenance.

The mechanisms for current provisioning techniques to handle varying workload demand may not scale for large-scale data processing. Nonetheless, one can admire the potential benefits of these techniques and argue that relevant mechanisms need to be developed for large data. Armbrust *et al.* (10) point out that there is a need to create a storage system that could harness the advantage of elastic resources provided by a cloud while meeting existing storage systems expectations in terms of data consistency, data persistence and performance.

Systems that jointly employ scheduling and provisioning have been explored in grids. The Falkon [Raicu *et al.* '07] scheduler triggers a provisioner component for host increase or decrease. This host variation has also been explored during the execution of a workload hence providing dynamic provisioning. Presently, tasks stage data from a data repository. Since this can become a bottleneck as data scales, scheduling exploiting data locality is suggested as a solution. The MyCluster project [Walker *et al.* '06] similarly allows Condor or SGE clusters to be overlaid on top of TeraGrid resources to provide a user with personal clusters. Various provisioning policies with different tradeoffs are explored including dynamic provisioning. The underlying motivation is to minimize wastage of resources. However, MyCluster is aimed at compute-intensive tasks. Given the similarities between grids and clouds, the joint techniques for scheduling and provisioning in these systems and related work are worth exploring for their relevance in clouds.

5. Conclusion and Future Research

Summary

The amount of data available for many areas is increasing faster than our ability to process and analyse it. The possibility of providing large number of computational resources by cloud computing has led to recent interest in exploiting clouds for large-scale data-intensive computing. However, data-intensive computing presents a set of new challenges, for systems management in a cloud, including the new processing frameworks, such as MapReduce, and the costs inherent with large data sets in a distributed environment. Workload management, an important component of systems management, is the discipline of effectively managing, controlling and monitoring "workflow" across computing systems. In this paper, we examine workload management for data-intensive computing in clouds.

We present a taxonomy for workload management of data-intensive computing in the cloud. At the top level of the taxonomy four main functions are identified: (a) workload characterization, (b) provisioning, (c)

scheduling and (d) monitoring. We focus on the scheduling and provisioning functions in the paper. The scheduling portion of the taxonomy categorizes scheduling methods in terms of several key properties such as (a) the scheduled work-unit, (b) the objective function optimized by the scheduling, (c) the scheme used to map work-units to resources, (d) the type of locality exploited in the scheduling, and (e) whether data replica management is integrated with scheduling or not. In examining current scheduling approaches we see that most systems consider entire workflows, optimize a user-centric objective like the makespan of a workflow, use simple dynamic mapping and exploit knowledge of replicas in placing tasks. Moreover, there is a need to better integrate scheduling and replica management and to balance global metrics such as cost with user metrics in scheduling.

The provisioning portion of the taxonomy categorizes provisioning methods in terms of (a) the kind of resource provisioned (physical or virtual), (b) the kind of trigger used to initiate provisioning (predictive, reactive or external), (c) and the scope of the provisioning (processing or storage). Note that systems employing provisioning use three methods namely scaling, migration and surge computing. Scaling is the primary method used in public clouds such as Amazon's EC2 where virtual processing images are automatically scaled using a reactive trigger based on user-defined conditions. Surge computing is used in the case of a hybrid private-public cloud combination. However, there is little work so far on the provisioning of storage resources.

Open Problems

Recent research on large-scale data processing, renewed interest in shared-nothing parallel DBMSs and the illusion of infinite resources offer exciting opportunities to process large amounts of data. Processing data on clouds poses a new tradeoff for data-intensive workload management. The tradeoff lies in the choices available for workload management in case of workload variation. In case of an increase in workload, the tradeoff for data processing has two choices. It could either multiplex the increased workload across existing resources or resize the resource pool to increase processing capacity. However, each of the choices has its own challenges. For example, multiplexing may lead to resource overload, QoS and deadline violations, increased response time and reduced fault tolerance. Increasing resources may address all of these issues but may require an access to data, data loading or rebalancing, and preprocessing. Furthermore, more resources on a public cloud mean additional (monetary) costs. Scaling in reduces cost but may require that data be off-loaded from unneeded resources.

A data-intensive workload manager in a cloud is a complex information management system, which has many tuning parameters for performance optimization. With integrating workload management features, a large number of threshold values to control workload are needed to be well understood and set by the cloud providers. This renders the entire system more complex in terms of operation and maintenance.

Estimating a system capacity plays an important role in the workload management process in DBMS, as all controls imposed on the users' requests are based on the system state. It is yet unclear how estimating a system capacity will turn out for a workload execution in a cloud with the resource pool scaling into hundreds and possibly more.

From the discussion presented in this paper, it is clear that there are several issues that need to be explored and addressed for a data-intensive workload management system in a cloud. It becomes more important when (a) systems need to automatically choose and apply appropriate techniques to manage users' workloads during execution, (b) dynamically estimating available system capacity and executing progress of a running workload, and (c) reducing the complexity of a workload management system's operation and maintenance. Second, provisioning of storage resources in a dynamic manner involves a number of problems including effective partitioning and replication of data, minimizing the impact of dynamic

reallocation of resources on executing work and finding new definitions of consistency appropriate for the cloud environment. Third, workload management methods that integrate scheduling and provisioning should be explored. The methods should be dynamic in order to fit into the elastic resource model of a cloud.

References

Abadi, D. J. (2009). Data Management in the Cloud: Limitations and Opportunities. IEEE Data Eng. Bull. **32**(1): 3-12.

Abouzeid, A., K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz and S. A. Rasin (2009). HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proc. VLDB Endow. **2**(1): 922-933.

Amazon. ('a). Elastic Compute Cloud (Amazon EC2). Retrived 17.9.11, from http://aws.amazon.com/ec2/.

Amazon. ('b). CloudWatch. Retrived 18.5.10, from http://aws.amazon.com/cloudwatch/.

Amazon. ('c). Auto Scaling. Retrived 18.5.10, from http://aws.amazon.com/autoscaling/.

Amazon. ('e). Elastic MapReduce. Retrived 19.5.10, from http://aws.amazon.com/elasticmapreduce/.

Amazon. ('h). Amazon Simple Storage Service (Amazon S3). Retrived 2.12.10, from http://aws.amazon.com/s3/.

Amazon. ('i). Amazon Elastic Block Store (EBS). Retrived 28.8.10, from http://aws.amazon.com/ebs/.

Amazon. ('k). Amazon SimpleDB (beta). Retrived 1.1.11, from http://aws.amazon.com/cloudwatch/.

Amazon. ('l). Washington Post. Case Study Retrived 18.2.11, from http://aws.amazon.com/solutions/case-studies/washington-post/.

Apache. ('a). Hadoop. Retrived 19.8.10, from http://hadoop.apache.org/.

Apache. ('b). Hadoop DBInputFormat. Retrived 14.12.10, from http://www.cloudera.com/blog/2009/03/database-access-with-hadoop/.

Apache. ('c). Hadoop Distributed File System. Retrived 19.8.10, from http://hadoop.apache.org/hdfs/.

Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia (2010). A view of cloud computing. Commun. ACM **53**(4): 50-58.

Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica and M. Zaharia (2009). Above the Clouds: A Berkeley View of Cloud Computing. Technical Report No. UCB/EECS-2009-28, University of California at Berkeley.

Bégin, M.-E. (2008). An egee comparative study: grids and clouds evolution or revolution? Enabling Grids for E-Science, CERN. **1**.

Brantner, M., D. Florescu, D. Graf, D. Kossmann and T. Kraska (2008). Building a database on S3. Proceedings of the 2008 ACM SIGMOD international conference on Management of data. Vancouver, Canada, ACM.

Brian, F. C., R. Raghu, S. Utkarsh, S. Adam, B. Philip, J. Hans-Arno, P. Nick, W. Daniel and Y. Ramana (2008). PNUTS: Yahoo!'s hosted data serving platform. Proc. VLDB Endow. **1**(2): 1277-1288.

Buyya, R., C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6): 599-616.

Casavant, T. L. and J. G. Kuhl (1988). A taxonomy of scheduling in general-purpose distributed computing systems. Software Engineering, IEEE Transactions on **14**(2): 141-154.

CERN. The Atlas Experiment. Retrived 10.7.10, from http://www.atlas.ch/fact-sheets-1-view.html.

Chaiken, R., B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver and J. Zhou (2008). SCOPE: easy and efficient parallel processing of massive data sets. Proc. VLDB Endow. **1**(2): 1265-1276.

Chang, F., J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes and R. E. Gruber (2008). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) **26**(2): 1-26.

Chappell, D. (2009). Introducing Windows Azure. Retrived 24.8.10, from http://download.microsoft.com/documents/uk/mediumbusiness/products/cloudonlinesoftware/IntroducingWindowsAzure.pdf.

Clark, C., K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt and A. Warfield (2005). Live migration of virtual machines. USENIX Association Proceedings of the 2nd Symposium on

Networked Systems Design & Implementation (NSDI '05). Berkeley, CA, USA, Usenix Assoc**:** 273-286.
Clark, S. (2000). Teradata NCR. Retrived 22.2.11, from http://www.teradata.com/library/pdf/butler_100101.pdf.
Cloudera. Hadoop training and support. Retrived 1.7.10, from http://www.cloudera.com/.
Dean, J. and S. Ghemawat (2004). MapReduce: simplified data processing on large clusters. Proceedings of the Sixth Symposium on Operating Systems Design and Implementation (OSDI'04): 137-149, Berkeley, CA, USA, USENIX Assoc.
Dean, J. and S. Ghemawat (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM **51**(1): 107-113.
Desprez, F. and A. Vernois (2006). Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid. Journal of Grid Computing **4**(1): 19-31.
Dewitt, D. and J. Gray (1992). Parallel database systems. The future of high performance database systems. Communications of the ACM **35**(6): 85-98.
DeWitt, D. J., E. Paulson, E. Robinson, J. Naughton, J. Royalty, S. Shankar and A. Krioukov (2008). Clustera: an integrated computation and data management system. Proc. VLDB Endow. **1**(1): 28-41.
Dong, F. (2009). Workflow Scheduling Algorithms in the Grid. School of Computing. Kingston, Queen's University. **PhD**.
Duncan, R. (1990). Survey of parallel computer architectures. Computer **23**(2): 5-16.
e2enetworks. demystifying-saas-paas-and-iaas. Retrived 20.2.11, from http://e2enetworks.com/2010/05/03/demystifying-saas-paas-and-iaas/.
Elmore, A., S. Das, D. Agrawal and A. E. Abbadi (2010). Who's Driving this Cloud? Towards Efficient Migration for Elastic and Autonomic Multitenant Databases. Tecnical Report 2010-05, UCSB CS.
Foster, I., Z. Yong, I. Raicu and S. Lu (2008). Cloud Computing and Grid Computing 360-Degree Compared. Grid Computing Environments Workshop, 2008. GCE '08: 1-10.
Garfinkel, S. L. (2007). An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS. Cambridge, MA., Harvard University.
Gilbert, S. and N. Lynch (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2): 51-59.
Google. App engine. Retrived 19.5.10, from http://code.google.com/intl/de-DE/appengine/.
Gray, J. (2008). Distributed Computing Economics. Queue **6**(3): 63-68.
GreenPlum. Greenplum Database Architecture. Retrived 19.8.10, from http://www.greenplum.com/technology/architecture/.
Grossman, R. L. and Y. Gu (2009). On the Varieties of Clouds for Data Intensive Computing. IEEE Data Engineering Bulletin **32**(1): 44-50.
Gruska, N. and P. Martin (2010). Integrating MapReduce and RDBMSs. Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '10), Toronto, Canada, IBM.
Gu, Y. and R. L. Grossman (2009). Sector and Sphere: the design and implementation of a high-performance data cloud. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **367**(1897): 2429-2445.
Gurd, J. R., C. C. Kirkham and I. Watson (1985). The Manchester prototype dataflow computer. Communications of the ACM **28**(1): 34-52.
Henderson, R. L. (1995). Job scheduling under the Portable Batch System: 279, Berlin, Germany, Springer-Verlag.
Hockauf, R., W. Karl, N. Leberecht, M. Oberhuber and M. Wagner (1998). Exploiting spatial and temporal locality of accesses: a new hardware-based monitoring approach for DSM systems. Lecture Notes in Computer Science: 206-215.
IBM. IBM Smart Cloud. Retrived 24.6.11, from http://www.ibm.com/cloud-computing/us/en/.
Isard, M., M. Budiu, Y. Yu, A. Birrell and D. Fetterly (2007). Dryad: distributed data-parallel programs from sequential building blocks. Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. Lisbon, Portugal, ACM. **26**.
JSSPP. Proceedings of Job Scheduling Strategies for Parallel Processing Workshop. Retrived 13.6.10, from http://www.link.springer.de/link/service/series/0558/tocs/t2221.htm.

Koop, M. J., W. Huang, K. Gopalakrishnan and D. K. Panda (2008). Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand. 16th IEEE Symposium on High Performance Interconnects: 85-92, Stanford, CA

Krompass, S., H. Kuno, J. L. Wiene, K. Wilkinson, U. Dayal and A. Kemper (2009). Managing long-running queries. Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT'09: 132-143, Saint Petersburg, Russia, Association for Computing Machinery.

Lagar-Cavilla, H. A., J. A. Whitney, A. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno and M. Satyanarayanan (2009). SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. Eurosys'09: Proceedings Of The Fourth Eurosys Conference. New York, Association for Computing Machinery**:** 1-12.

Lagar-Cavilla, H. A., J. A. Whitney, A. Scannell, R. B. P. Patchin, S. M. Rumble, E. d. Lara, M. Brudno and M. Satyanarayanan (2010). SnowFlock: Virtual Machine Cloning as a First Class Cloud Primitive. ACM Transactions on Computer Systems (TOCS) **19**(1).

Lim, H. C., S. Babu and J. S. Chase (2010). Automated control for elastic storage. Proceeding of the 7th International Conference on Autonomic Computing, ICAC '10 and Co-located Workshops: 1-10, Washington, DC, United states, Association for Computing Machinery.

Litzkow, M. J., M. Livny and M. W. Mutka (1988). Condor-a hunter of idle workstations. 8th International Conference on Distributed Computing Systems.: 104.

Luo, Y., B. Zhang, X. Wang, Z. Wang, Y. Sun and H. Chen (2008). Live and incremental whole-system migration of virtual machines using block-bitmap. Cluster Computing, 2008 IEEE International Conference on: 99-106.

Madden, S., D. DeWitt and M. Stonebraker. (2007). Database parallelism choices greatly impact scalability. DatabaseColumn Blog. Retrived 8.5.10, 2010, from http://databasecolumn.vertica.com/database-architecture/database-parallelism-choices-greatly-impact-scalability/

McKinley, K. S., S. Carr and C.-W. Tseng (1996). Improving data locality with loop transformations. ACM Trans. Program. Lang. Syst. **18**(4): 424-453.

Mehta, A., C. Gupta, S. Wang and U. Dayal (2008). Automatic Workload Management for Enterprise Data Warehouses. IEEE Data Eng. Bull. **31**(1): 11-19.

Mian, R., P. Martin, A. Brown and M. Zhang (2010). Managing Data-Intensive Workloads in a Cloud (poster). Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '10), Toronto, Canada, IBM.

Mian, R., P. Martin, A. Brown and M. Zhang (2011). Managing Data-Intensive Workloads in a Cloud. Grid and Cloud Database Management. G. Aloisio and S. Fiore. Heidelberg, Springer.

Moore, R., T. A. Prince and M. Ellisman (1998). Data-intensive computing and digital libraries. Commun. ACM **41**(11): 56-62.

Moreno-Vozmediano, R., R. S. Montero and I. M. Llorente (2009). Elastic management of cluster-based services in the cloud. Proceedings of the 1st workshop on Automated control for datacenters and clouds (ACDC). Barcelona, Spain, ACM**:** 19-24.

Niu, B., P. Martin and W. Powley (2009). Towards autonomic workload management in DBMSs. Journal of Database Management **20**(3): 1-17.

Olston, C., B. Reed, U. Srivastava, R. Kumar and A. Tomkins (2008). Pig latin: a not-so-foreign language for data processing. Proceedings of the 2008 ACM SIGMOD international conference on Management of data. Vancouver, Canada, ACM.

Prodan, R. and S. Ostermann (2009). A survey and taxonomy of infrastructure as a service and web hosting cloud providers. Grid Computing, 2009 10th IEEE/ACM International Conference on: 17-25, Banff, Canada.

Quiroz, A., H. Kim, M. Parashar, N. Gnanasambandam and N. Sharma (2009). Towards autonomic workload provisioning for enterprise grids and clouds. 2009 10th IEEE/ACM International Conference on Grid Computing (GRID): 50-57, Banff, AB, Canada, IEEE Computer Society.

Raicu, I., I. Foster, A. Szalay and G. Turcu (2006). AstroPortal: A Science Gateway for Large-scale Astronomy Data Analysis. TeraGrid Conference.

Raicu, I., Y. Zhao, C. Dumitrescu, I. Foster and M. Wilde (2007). Falkon: a Fast and Light-weight tasK executiON framework. Proceedings of the 2007 ACM/IEEE conference on Supercomputing. Reno, Nevada, ACM.

Ranganathan, K. and I. Foster (2002). Decoupling computation and data scheduling in distributed data-intensive applications. Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing: 352-358, Piscataway, NJ, USA, IEEE Comput. Soc.

Sanjay, G., G. Howard and L. Shun-Tak (2003). The Google file system. SIGOPS Oper. Syst. Rev. **37**(5): 29-43.

SDSS, S. D. S. S. Mapping the Universe. Retrived 15.12.10, from http://www.sdss.org/.

Shatdal, A., C. Kant and J. F. Naughton (1994). Cache conscious algorithms for relational query processing: 510, Santiago, Chile, Morgan Kaufmann Publ Inc.

Sotomayor, B., R. S. Montero, I. M. Llorente and I. Foster (2009). Virtual infrastructure management in private and hybrid clouds. IEEE Internet Computing **13**(5): 14-22.

Stonebraker, M., S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem and P. Helland (2007). The end of an architectural era: (it's time for a complete rewrite). Proceedings of the 33rd international conference on Very large data bases. Vienna, Austria, VLDB Endowment.

Sybase. (2007). Sybase IQ Powers World's Largest Green Data Warehouse Including Unstructured Data. Retrived 4.5.11, from http://www.sybase.com/detail?id=1054047.

Thain, D., T. Tannenbaum and M. Livny (2005). Distributed computing in practice: the Condor experience. Concurrency And Computation-Practice & Experience **17**(2-4): 323-356.

Thusoo, A., J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff and R. Murthy (2009). Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow. **2**(2): 1626-1629.

Vaquero, L. M., L. Rodero-Merino, J. Caceres and M. Lindner (2008). A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. **39**(1): 50-55.

Venugopal, S., R. Buyya and K. Ramamohanarao (2006). A taxonomy of Data Grids for distributed data sharing, management, and processing. ACM Comput. Surv. (CSUR) **38**(1): 123-175.

Vertica. (2009). Vertica Analytic Database for the Cloud gets an upgrade. Retrived 22.2.11, from http://www.vertica.com/news/press/vertica-analytic-database-for-the-cloud-gets-an-upgrade/.

Voorsluys, W., J. Broberg, S. Venugopal and R. Buyya (2009). Cost of virtual machine live migration in clouds: A performance evaluation. 1st International Conference on Cloud Computing **5931 LNCS**: 254-265, Beijing, China, Springer Verlag.

Walker, E., J. P. Gardner, V. Litvin and E. L. Turner (2006). Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment **2006**: 95-103, Paris, France, Inst. of Elec. and Elec. Eng. Computer Society.

Weissman, C. D. and S. Bobrowski (2009). The design of the force.com multitenant internet application development platform. Proceedings of the 35th SIGMOD international conference on Management of data. Providence, Rhode Island, USA, ACM.

Yu, J. and R. Buyya (2005). A taxonomy of scientific workflow systems for Grid computing. Sigmod Record **34**(3): 44-49.

Zaharia, M., D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker and I. Stoica (2009). Job Scheduling for Multi-User MapReduce Clusters, Electrical Engineering and Computer Sciences, University of California at Berkeley.

Zhang, H., G. Jiang, K. Yoshihira, H. Chen and A. Saxena (2009). Resilient workload manager: Taming bursty workload of scaling internet applications. 6th International Conference on Autonomic Computing, ICAC'09: 19-28, Barcelona, Spain, Association for Computing Machinery.

Zhang, M., B. Niu, P. Martin, W. Powley, P. Bird and K. McDonald (2011). Utility Function-based Workload Management for DBMSs. Proceedings of the 7th International Conference on Autonomic and Autonomous Systems (ICAS 2011): 116-121, Mestre, Italy.

Zhou, S. N., X. H. Zheng, J. W. Wang and P. Delisle (1993). Utopia - A Load Sharing Facility For Large, Heterogeneous Distributed Computer-Systems. Software-Practice & Experience **23**(12): 1305-1336.