Adaptability and Caching in Component-based System Environment

Omar Asad
School of Computer Science
McGill University
oasad@cs.mcgill.ca
August 2012

Abstract. The rapid growth of Internet-based applications in terms of users, data size, and physical resources, raises a great challenge to service providers and system administrators to maintain a certain level of Quality of Service (QoS) while at the same time maximizing resource utilization. In addition, web-enterprise emerging technologies, such as cloud computing and virtual machines, require a new trend of hosting several applications within the same super machine, which, in turn, needs an insightful analysis of the existing applications' behaviours to accommodate each of them with sufficient resource to work properly. Also, such growth still needs to exploit legacy technologies, specifically data caching, to enhance system performance and achieve customer satisfaction. This survey presents the latest research and industry work that has been carried out in the areas of system adaptability and data caching in distributed systems. More specifically, system adaptability answers the question of how a system reacts to a fluctuating workload it receives so as to maintain its QoS. Another aspect is to do autonomic provisioning for a certain system by figuring out the resources it needs to handle different workload situations. Caching, on the other hand, works by storing data that was previously served in main memory with a faster access time than that of the hard drive. Thus, physically caching the data and partitioning the cache between different coupled applications are important issues. Lastly, in talking about both technologies (i.e. adaptability and caching) we have to consider large scale database processing in a cloud environment.
Keywords: Distributed Systems, Autonomic Provisioning, Caching, Transactions.

1 Introduction

In the past few years, use of the Internet has changed dramatically because of the technology integration in almost every aspect of human life. This use includes on-line shopping, social networking, gaming, medical treatments, and email. As an effect, both researchers and industry pioneers have started to accommodate new technologies such as cloud computing and virtual machines (VM) to host the large numbers of enterprise applications in order to provide reliable service to users. However, for the service provider, the challenge becomes how to manage these applications while maintaining a certain level of Quality of Service (QoS).

The above challenge of managing the applications can be treated in different ways according to the distribution of the system components; if there are several machines dedicated to host one system component, then the adaptation could be to allocate extra machines in the peak workload time while deallocating some machines in the low demand time [8, 18, 17, 6]. Alternatively, several applications could be hosted by a super capacity server. In this case, the question becomes how to distribute the physical resources among the different applications according to

their workload variations [15, 10, 14, 16]. Even within each application, there is a space to control the service level according to the workload against different resources (CPU, memory).[13, 5].

A common and efficient way of enhancing the system service and reducing the response time is caching, where the objects are accessed locally at a front-end server instead of querying the back-end server which is more costly. Several caching architectures and techniques have been discussed in the literature [7, 12, 2, 3], including query cache where the result of a previous query is stored in the cache. However, when considering a replicated system, the consistency between different cache locations as well as between the cache and the database has to be taken into account. Also, in a shared physical resources environment that is enabled to host several applications together, it is important to partition the cache memory among different shared applications to achieve optimal performance[16].

Furthermore, due to emerging cloud computing, several cloud database systems have been presented [1, 19, 11, 4], which can provide database as a platform for cloud customers. Some of these newly database systems, have customized the traditional transaction management to adopt to the cloud nature and to maintain the ACID properties in the underlying storage system.

In general, this survey covers a broad area of distributing systems, ranging from system self-adaptation to data caching to large scale database platforms for cloud. The rest of this paper is organised as follows: Section 2 provides the background in component-based systems and introduces some terminology that is frequently used throughout the paper. Section 3 presents adaptability in distributed systems and how the system is tuned to the different variations of workload. Section 4 examines caching and distributing caching memory among several applications. Section 5 discusses database platforms as a service in cloud computing. Section 6 concludes the paper.

2 Background

Since this survey contains several topics on distributed systems, it becomes important to provide an overview about the technologies that are used, in addition to some vital definitions of the terminology that are mentioned frequently throughout this report.

2.1 Enterprise Application

Most businesses nowadays are using technologies and software to automate the entire process of their work, store the data, and manage it in an efficient and accurate manner. Moreover, several businesses are marketing and selling their products through the Web, which is known as e-commerce. The way these enterprise systems are organized and function follows distributed system convention, in which the system components are distributed logically and physically through different tiers to achieve scalability, availability, and robustness against failures. It is also worth mentioning that although enterprise applications are the major sector of Web residents, there are also plenty of other domains hosting their applications and software on the Web to provide easier accessibility for their users. Such applications include portal systems, news, and scientific data.

2.2 Multi-tier Architecture

Enterprise software applications are divided into the following tiers: presentation tier, logic tier, and data tier. The presentation tier handles the client requests and generates the reply, while the logic tier deals with processing the information and managing the interactions between the presentation tier and the data tier. The data tier normally hosts the application data, provides the logic tier with the requested data, and processes data updates. Each tier is logically separated from the others; and, in most scenarios, each tier is hosted on a dedicated server or cluster of servers. Therefore, this architecture facilitates deployment and gives more flexibility in designing each tier without affecting the other tiers. Also, it provides more robustness against failure by replicating each tier in more than one machine.

2.3 Transactions

A transaction is used to describe the process of user interactions with the system. Generally, a transaction is defined as a set of operations in which all are treated as a one logical unit. Transaction execution is described by four properties: Atomicity, Consistency, Isolation, Durability or ACID in short. Atomicity requires each transaction to be executed as one unit. At the end, all of its operations succeed or the transaction does not leave any effect in the database. Consistency states that the transaction execution will move the system from one stable state to another stable state. Isolation guarantees the separation of concurrent executing. Durability means that a committed transaction cannot be rolled back. Thus, the changes done by a transaction become permanent.

Monitoring the Quality of Service (QoS) of transaction execution is an important factor to evaluate the system's performance. Most work in a distributed system uses two metrics to express the system's QoS: response time and throughput. Response time defines the time needed for a transaction to be executed. It starts when the client issues a request to the system until the response is received back. Throughput express the transaction execution rate, or simply the number of transactions the system can process per time unit.

2.4 Concurrency Control

To achieve the isolation property of transactions, a concurrency control mechanism is used. One widely known mechanism is snapshot isolation, which is a multi-version database concurrency control model. When a transaction begins, it receives a logical copy (snapshot) of the database that is the last committed state. This version is not affected by other transactions and all the changes the transaction performs on that snapshot are visible only to the current transaction before commit. A read-only transaction can commit directly after finishing read from its snapshot. However, an update transaction commits only if there is no write-write conflict with any other concurrent committed transactions. Once the transaction commits, it produces a new version of the database and all the changes made become visible to newly started transactions.

2.5 Cloud Computing and Virtual Machines

Traditionally, cloud computing was used to refer to the Internet. Nowadays it defines the computing resources which are being deployed over the Internet in order to enable the application's

owners to exploit these resources and pay for their use. This technology enables the computing consumers to invest in their applications by adding new capacities on demand, without investing in new hardware, software platforms, or training new stuff. Cloud computing consists of a hardware layer which is a set of server clusters that run a virtual operating system on top of them. Thus, these clusters are logically divided through virtual machines (VM).

A VM is the lowest level software abstraction that runs on top of a minimal operating system. It is a software that can deploy and run programs like any standalone operating system. Several VMs can be installed on the same server with a solid isolation between each of them, where the underlying physical resources have to be managed and allocated to each VM depending either on demand or as a fixed quota.

The cloud customers are offered any of the following services:

- Infrastructure as a Service (IaaS), where a unit of computer hardware is offered as a virtual machine.
- Platform as a Service (PaaS), where not only hardware resources but also a development and runtime environment is offered, such as programming language execution environment, database, and Web server.
- Software as a Service (SaaS); by allowing the customers to consume an application over the Internet instead of installing it locally and eliminating the need to install and run the application as well as simplifying system support.

In general, the cloud providers use data centres to host their resources and provide easy access to them. A data centre is a facility that houses computer machines and associated components, such as storage system, backup devices, and routers. Normally, a data centre contains racks of servers where each can run a standalone operating system and is dedicated to one resource-hungry application or, at the other extreme, hosts several VMs and each VM will be allocated to one small to medium application.

2.6 Replication

Replication is a widely used approach for data-intensive applications to distribute the workload equally among different replicas in order to achieve higher performance and at the same time provide a robustness against failure by letting other replicas take over once a replica fails.

In a multi-tier architecture, any tier could be replicated. However, database replication is a widely known approach that consists of a set of database servers each having a copy of the full database. database replication can be classified based on whether all replicas are able to accept update transactions that change the database state. In a multi-master approach, every replica is able to perform any read or write operation and is responsible for propagating any write operation to the other replicas. On the other hand, a single-master approach requires the write operations to be submitted and executed exclusively in the master replica, which then propagates them to other replicas, while any replica can process the read operation. In general, the multi-master approach is more flexible, but it needs a solid concurrency control mechanism to isolate concurrent transactions on different replicas. In a single master approach concurrency control is easier, since all concurrent update transactions can be detected at the single master.

2.7 Distributed Systems' Adaptability

Distributed system performance is heavily affected by the workload the system handles. While the service providers aim to maintain a certain level of QoS, it becomes a non-trivial task to guarantee such a level at peak service demand times. Adaptability is a new concept that aims to enable the system to dynamically manage itself, continuously optimize its performance, detect failures and try to fix them [18]. Thus, the goal of adaptability is to minimize human involvement in managing the system as well as to enhance its performance.

2.8 Caching

Caching increases the level of system performance by storing the data that was previously accessed by clients in the front-end server. So, for subsequent requests, the local cache will be checked to see if the associated data exists; if the data is stored in the cache, then the front-end server returns it back to the client; otherwise, the request will be sent to the back-end server. Caching enhances the performance by lowering the client response time and minimizing the back-end bottleneck probability on peak demand.

3 Adaptability

Research on distributed systems' adaptability tries to enhance system performance and provides the system with the capability to take necessary ad-hoc actions in order to maintain its availability and scalability. Such actions include distributing load between different servers, allocating or deallocating physical resources, and upgrading or degrading their service. At the beginning, we will discuss load balancing, then resource allocation, and finally service level controlling.

3.1 Load Balancing in Replicated Database

In a replicated database, the load balancer routes the transactions to the different replicas in a way that reduces the replica overload and at the same time achieves a faster query execution than a standalone database. Different load balancing strategies can be used, such as round robin or least active connections(route the request to the server which has least number of connection), which are able to balance the load well, but they can lead to poor performance by causing memory contention.

Elnikety et al [9] develop a memory-aware load balancing which works by distributing the transactions among replicated databases in a way that avoids memory overhead and reduces disk I/O. To achieve this goal, the authors assume a database application with a predefined set of transaction types; the goal becomes how to group different transaction types and then allocate each group to a set of database replicas in a way that allows transaction processing to occur in the local memory and, hence, reduce the disk I/O operations.

Transaction grouping can be done based on their database working set (tables and indices accessed by a particular transaction). Several grouping techniques were introduced that exploit the transaction information:

1. Transaction type, like placingOrder, browseCategories, browseRegions
2. Working set size, which represents the size of the tables and indexes used by a particular transaction type
3. Working set content, by knowing which tables and indexes are used by a specific transaction type.
4. Working set access pattern, which takes into account the access pattern of query execution.

After getting the estimated working set information, transaction types can be grouped using different methods:

1. Memory Aware Load Balancing-Size only (MALB-S), which groups transaction types so that the sum of all working set sizes fits into the memory of one replica. Overlap of working sets is not considered. For example, suppose there are two transaction types T1 and T2. T1 uses tables A and B, while transaction T2 uses tables B and C. Then the estimated memory size needed for both transactions is A+B+B+C.
2. MALB-SC (Size+Content), the same as MALB-S but it avoids re-counting the shared (overlapped) content. So in the previous example, the total table size is A+B+C.
3. MALP-SCAP(Size +Content Access Pattern), the same as MALB-SC but it has extra knowledge of linearly scanned tables.

At the start of execution, each group gets allocated a set of replicas that will process transactions of this group. To dynamically reallocate, the loadbalancer calculates the load for each group by averaging the CPU and disk utilization for all replicas assigned to that group. Then it compares the loads of the different groups and allocates extra replicas to the most loaded group by borrowing them from the least loaded one. Moreover, there is a possibility to re-allocate more than one replica at a time in order to avoid poor performance.

To further enhance the performance, an update filter approach was used. The update filtering allows the tables that were not used by a certain replica to be outdated. Only replicas that access a certain table will receive all updates on this table as they occur. Hence, the overhead of immediately reflecting updates on all replicas is reduced.

3.2 Dynamic Physical Resources Allocation

In a dynamic Web system with fluctuating workloads, the static provisioning of allocating machines to each application scales poorly and causes underutilization of resources, since the load varies over time. Thus dynamic resource allocation aims at dynamic reallocation of machines to each application over time according to the current workloads of the different applications. The task of allocating or deallocating machines to a certain application is done by a Resource Manager (RM) that receives performance feedback periodically from different machines; and based on it, it makes the proper resource allocation decision and multicasts it to the participated machines.

3.2.1 Dynamic Machines Allocation In [17, 18], the authors exploit a Reinforcement Learning (RL) approach to adapt the system to the optimal resource allocation decision. RL

can be defined as the process of learning in an arbitrary environment by taking an action and noticing the consequences of that action. The intention is to generate a policy that will optimize the long term goal.

In particular, the problem was treated as a Markovian Decision Process (MDP), which is a mathematical optimization framework. MDP represents a system that consists of different states and, at a certain time, chooses to move to the next state by taking a certain action and getting an immediate reward. To learn the MDP policy, the authors utilize the SARSA method that runs in association with each application.

Once the system starts, each application receives the number of allocated machines from the RM and registers the average request demand. In turn, the application monitors the average response time, which is used to express the reward value. Therefore, a low response time receives a high reward, while a high response time receives a negative reward. At each step, the application updates its own value function table, which is a two-dimensional array; each entry on it represents the result of a SARSA function with different configurations of the number of machines and average request demand.

Periodically, the RM requests the value function from each application, and uses it to compute the optimal resource allocation. The RM simply maximises the summation of value functions by joining all possible allocations and picking the allocation policy that achieves the highest global reward, and sends it back to the different applications.

3.2.2 Dynamic Backend Database Provisioning Not unlike dynamic resource allocation, dynamic service provisioning aims to determine the systems' physical needs to maintain a certain level of QoS according to fluctuating workloads. An analytical model to predict the response time as well as the system throughput for a replicated database from a stand-alone database is presented in [8].

Basically, the model assumes that a given number of clients submit a given number of transactions per second to a given number of replicas. The transactions are a mixture of read and write operations with a fixed ratio between them. Furthermore, the system is modelled as a closed-queueing network in which the CPU and disk of each database replica are considered as a service centre with a queue. The service times are taken by measurements from a standalone database. Other tasks such as communication, load-balancing, and certification are modelled as a constant delay.

Since the replicated database could be either a single master approach or a multi-master approach, a solution for each of them is proposed based on a snapshot isolation concurrency control. The model first estimates the time needed for each transaction in a standalone database system by finding the required service time for both transaction types: read-only and update transactions. For update transaction, there is a probability to abort if conflict occurs. Thus, the model takes into consideration the abortion probability and finds the average service time.

After that, the model is extended to cover a multi-master replicated database. The cost model is the same as the standalone approach, except it considers the cost for propagating write sets between replicas. Also, the single master model is almost the same as the multi-master but it is more complex, since all read transactions have to be executed in the slave

nodes while the update transactions have to be forwarded to the master node, and then the changes have to be propagated to the slaves.

Instead of predicting the database scalability in advance as described in [8], the authors in [6] describe a dynamic provisioning of database replicas depending on the application demand.

There are two ways to determine the number of replicas needed. One is reactive provisioning where the RM allocates or removes replicas from the system depending on the previous system performance.

The other is proactive, where the RM dynamically predicts the future state of the system and determines whether to add or remove replicas. More specifically, the RM predicts the status of the application for the next time period and classifies it into two categories: the status is within an acceptable response time range or the range is violated. If the predicted future response time is going to be above a certain threshold, then another replica will be added. Otherwise, if the response time is below a certain threshold, the RM triggers a replica removal.

Since replica addition is costly owing to data migration and system instability, the RM enters a tentative removal state after removing some replicas. In such a state, the replica continues to be updated but it is not used for load balancing read quires. After a certain time, if the average response time is still under the low threshold, then the replica is removed.

Another way of stabilizing the Web enterprise servers' performance under a heavy workload is by controlling the service level on the resource-intensive components. The service level adaptation can be found in many Web applications; for example, limiting the size and precision of a search result, approximating sort result, and resolution of video streaming.

In [13], Philippe et al introduce a dynamic approach where an RM controls the level of service by upgrading it when there is low load and degrading it in an over load. After the adaptation has been applied, the RM measures the impact of adaptation on resource usage for improving decision making in the future.

3.3 Dynamic VMs' configuration

VMs enable multiple operating systems to share resources on the same machine. Each VM hosts a different application. In such an environment, distributing the physical resources, such as CPU and memory, among different VMs can be done statically in advance, which might cause poor performance and reduced resource utilization. Therefore, dynamic VM configuration, which automatically redistributes the resources between different VMs based on their workload variation, can enhance the system's performance and cause better overall resource utilization.

In the cloud environment, some customers might have small to medium-size applications which do not require a standalone machine to host each of them. Therefore, the cloud infrastructure is designed in such a way that all the hardware resources are pooled in a shared infrastructure and applications share these resources as their demands are changing over time. Considering the previous scenario, where a certain cloud provider hosts multi-tier applications, the challenge becomes to propose an adaptive RM that has the ability to dynamically adjust the resource share in different tiers in order to meet the QoS and achieve maximum resource utilization.

The authors in [10] implement a testbed where multiple multi-tier applications share a common pool of physical resources. Basically, two tiers were considered for each application, the application tier and the database tier. Each tier is hosted in a separate VM and all the VMs of the same tier reside on the same physical machine.

The RM design, which acts as a feed-back loop, considers the system as an input-output model. The system inputs consist of entitlement (CPU share), which represents the CPU percentages for both the application tier and the database tier. The output contains the CPU utilization, which represents the ratio between the entitlement and the actual CPU usage in addition to the QoS metrics, such as response time and throughput.

To find the relation between the input and output parameters, an experimental testbed is set up which models a single mutli-tier application and multiple mutli-tier applications. For a single mutli-tier application, a single testbed node or RUBiS benchmark was used, which generates a certain workload of clients against the application. At the same time, the CPU entitlement was changed during the experiment for both the application and database tiers. After several experiments, a model that governs the relation between the given entitlement and the desired response time was formulated.

The same problem of VM auto-configuration is examined in [14]. The authors exploit a RL approach to automatically adjust a shared resource among several VMs during runtime.

To formulate the VM configuration problem as a RL task, the authors match the RL learning parameters into a VM problem as follows: the reward function is measured by a score that represents the ratio between the current throughput to a reference predefined throughput plus some penalties when the response time violates a predefined value. The state space is defined by the set of configuration parameters that affect the performance of each VM. Those parameters are memory size, scheduler credits, CPU time. For each of the three previous configurable parameters, there are three possible actions: increase, decrease, nop (no action).

The RM manages the VM configuration by monitoring the system feedback, which is sent to it in regular periods. After receiving feedback, the RM chooses an action that maximizes the total amount of reward and sends it to the different VMs.

Instead of adapting VMs that host application servers, Soror et al [15] posit different DBMS. Each runs on a single virtual machine and handles a varying workload. In this scenario, the RM design objective is to reduce the total resource consumption.

The cost model for each database server is challenging for two reasons. First, different database servers use different cost models. So, for example, one could define the cost model as the query execution time, while another could assume total resource consumption. Therefore its assumed that all database servers define the cost as total resource consumption. Second, the cost estimation problem depends on the database server configurations' parameters, but the suggested RM is given a candidate resource allocation. Therefore a mapping technique was used which maps a candidate resource allocation to a set of DBMS configuration parameter values.

4 Caching in Web Enterprise Systems

Caching is a widely known technique for enhancing system performance by decreasing client response time and reducing the number of quires accessing a back-end server. Recent research on caching is extended to cover cloud computing and partitioning the cache objects among several machines. However, the main concern is still data consistency between different front-end servers as well as between the front-end server and the back-end data server.

4.1 Caching Architecture and Protocols

In [2], Attar and Ozsu present three alternative caching architectures for a two-tier architecture with application server and database server. The proposed architectures are the following:

1. No caching or replication, in this scenario there is only one application server, and therefore, there is no caching or consistency problem.
2. Replicating static data and application code, where a proxy is composed in each site that contains the code and the static data and all the dynamic requests are served in the database server. Thus, strong consistency is automatically provided.
3. Replicating static data, application code and dynamic data. This is similar to the second case except that the dynamic content is also cached in each application server. Whenever a requested query result is cached in the local front-end server, then the query is answered without having to access the back-end server. Otherwise, the query is sent to the back-end server and the answered result is cached in the local server.

In case 3, each application has a local database that acts as cache storage for the local server and, at the same time, contains the same tables as the original DBMS in addition to two extra columns in each table: validity to indicate if the specified row is valid, or it has been invalidated by an update operation. The second column includes counters which keep track of each row's access frequency.

The query result is decomposed and stored in its corresponding local table. If the row already exists, its counter increases by one, otherwise the counter is set to 1.

Query result caching is done by maintaining two data structures in the memory of each application server:

1. Proxy-Side Cache Descriptor (PCD): list of elements containing pairs of query types and the parameter values and corresponding read locks that represent the read lock in that query
2. Server-Side Cache Descriptor (SCD): the same as the above; but instead of the read lock element, this structure contains the others server's cache elements.

There are several choices of where to place the cache location. One is to locate it in the application front-end server, or it can be in the database back-end server, or it could reside in a separate machine. It is obvious that the more upfront the cache is, the shorter latency the hit will take. However, if the cache resides in the application server, the throughput may suffer in case of server bottleneck.

4.2 Transparent Caching

In [3], Amza et al. study transparent caching, which presents a logical way of placing the cache between the application server and the back-end database server. Thus, the cache will appear to the application as a database whereas it looks an application to the database server.

The cache data structure is composed of a large hash table of cache entries. Each entry contains basically the SQL query string and the corresponding result. Once a Web application receives a client read requests, it checks the cache. If the result is present in the hash table, it sends it back. Otherwise, it forwards the request to the database and caches the returned results. In case of a write request, the cache invalidates all the cache entries related to that query and forwards the request directly to the database to be executed.

4.2.1 Partial Scheme To increase caching efficiency, the authors in [3] propose a partitioning scheme for insert queries. In this case, the cache will create a temp table that maps each table in the database, so that every row in the temp table will be the same as its original database table. Once the insert statement arrives at the cache, it is executed in the corresponding temp table instead of being executed in the database. Upon receiving a select query, it will be executed in the original database table in addition to an extra query that will be executed in the temp table. The final result will be finalized by merging the two query results. However, to guarantee consistency, the temp table entries need to be inserted into the original database table after a certain number of rows. The advantage of this scenario is to avoid invalidating the cache entry every time a new item is inserted in the table. Thus, the cache entries can be reused more often.

4.2.2 Detecting Coverage Another strategy to enhance cache performance is to detect whether a query results is fully presented (coverage) in the cache or only partially presented. Partial coverage means that part of the results of the requested query are already present in the cache due to a previous query with more restrictive parameters. In case of full coverage, the results will be returned to the client. If the query is partially covered, then instead of computing the whole query again at the database, the algorithm checks the remaining results of the query and forms a new query to send to the database. The final result will be achieved by merging the results from the new query and the partial results already cached.

Consistency is guaranteed by using Invalidation. Invalidation is a known method to guarantee consistency whereby the write operations invalidate the cached objects. In more detail, consider multiple caches, each resides in one server. Once any of the caches receives a write query, it will forward it to the database; and, at the same time, the cache processor extracts its accessed objects in order to invalidate them at the local cache as well as at the remote caches. Once the local cache receives result from the database and the acknowledgements from other caches, it sends response back to the client.

4.3 Consistent and Scalable Cache Replication

The application server replication in an enterprise's multi-tier architecture enhances performance of the system and increases its reliability. Moreover, using a local cache in each server

could decrease accessing the database frequently and, thus, lessen the client response time. But in such scenario the database is still a bottleneck and the system is more vulnerable to crashes because it is a one-point failure. The authors in [12] resolve these problems by grouping all of the application servers, a local cache and a local database at the same replica. Therefore, all replicas are communicating with each other to maintain consistency using a group communication system, in which there is no shared database among the replicas.

Since application server and database sever are grouped within one machine, a transaction should be executed in the cache as well as the local database. The transaction commit has to take place in the local cache, local database, remote caches, and remote databases. In order to do so, the authors present a replicated cache protocol that depends on snapshot isolation at the cache level as well as the database level. Once a transaction has committed at one site, its changes are propagated to all other replicas, including the local one using total order multicast. Therefore, the permanent changes will be consistent in all replicas because they all process the writes in the same order.

To guarantee the correct snapshot is served and facilitate concurrency control, a protocol to maintain versions for each object in the cache was proposed. The version is checked every time a transaction has to commit to see if a particular object is being accessed by another transaction at the time the transaction was being processed. Once the transaction is committed, the modified version becomes global and accessible by other transactions.

4.4 Partitioning of the Cache Hierarchy

Large data centres deploy several DBMSs on the same machine that are connected with a farm of storage devices at a lower level. The storage server is used to store the data blocks for different DBMSs. In this design there will be two caches: One attached to the DBMS, which is known as a buffer pool, and the second attached to the storage server.

Partitioning the cache in both locations among several hosted DBMSs is discussed in [16]. The general goal of the work is to find a partitioning caching policy (the buffer pool quota as well as the storage cache quota for each DBMS) that will maximize the utilization for all the DBMSs. The problem can be solved through the following steps:

1. Learn a function of the caching policy for each application. The input is caching policy and the output is the response time.
2. Map the response time to the corresponding utility value, which is a predefined function of the operator set to represent the QoS.
3. Solve the problem of finding the global optimal caching policy for all the applications.

5 Database Cloud

Cloud computing providers can pool database resources and offer customers a database-as-a-service model in the form of a database cloud. The customer data could be deployed as a platform consolidation, where multiple databases share the same physical resources. Or, on the other hand, it could be deployed as a database consolidation, where different database schemas

reside on a single database. However, the new cloud storage systems such as, Google BigTables and Amazon SimpleDB, do not guarantee the ACID properties. Moreover, the level of accuracy needed in processing a large amount of data depends on the type of processing. If it is a client transaction, then it has to be done fast and must provide the most recent data. While large data analytical transactions could accept long response times and sacrifice some level of data accuracy. This section will cover some work that has been carried out to elevate the service level of the cloud data storage as well as to achieve data processing accuracy. More specifically, four systems will be introduced: CloudTPS, HadoopDB, ES2 ,and Percolator.

5.1 CloudTPS

Many new cloud computing storage systems like Amazon DB and Google Bigtable do not guarantee strong consistency. For instance, the application may read stale data. Moreover, the mentioned storage systems do not apply transactions on multiple tables. On the other hand, they are able to provide a scalable and highly available cloud data store. CloudTPS[19] resolves the above limitations by providing a transactional model for distributed applications in the cloud, where cloud data storage is able to perform the ACID transactions and, thus, maintain data consistency even in the presence of failure.

The design model of CloudTPS works within the context of mutli-tier architecture, where there is a client accessing a cloud Web application, which in turns communicates with the back end data storage through a Transaction Processing System (TPS).

TPS consists of a number of Local Transaction Managers (LTM), each of which is responsible for a part of the data items. Once the system starts, the data is loaded from the cloud storage system into the TPS layer, then each LTM is assigned a subset of the loaded data. Fault tolerance is achieved by replicating the LTMs with their data into several nodes.

The client request is handled first by Web application, which in turn issues a transaction to a TPS. Since TPS contains several LTMs, the incoming transaction is assigned to any LTM that contains a subset of data accessed by that transaction. That LTM becomes the transaction coordinator across all LTMs. The transaction is then executed and, at the end, a 2PC protocol is used to commit or abort the transaction.

5.2 HadoopDB

MapReduce is a programming model that is capable of processing a large amount of data in parallel. MapReduce distributes the working data set among several nodes(this step is known as Map), and then gathers the results from different nodes into one output (reduce).

In [1], Abouzeid et al. present HadoopDB, which exploits the MapReduce programming model by extending it to accommodate DBMS. The basic idea behind this procedure is to connect multiple nodes of a DBMS cluster that utilizes Hadoop as a communication layer and a task coordinator. Thus, SQL queries are distributed and executed in parallel among different nodes using the MapReduce framework, while, at the same time, trying to push the query to be executed to local node instead of transferring it to another node to achieve higher performance. In general, HadoopDB capable of utilizing the power of individual DBMSs and, at the same time, boosting the performance more than other data stores.

5.3 ES2

A typical cloud database system has to deal with different kinds of access to data stored in the back-end data storage. One kind is online updates initiated by the system's clients and known as Online Transaction Processing (OLTP), and another is periodic large-scale analytical processing known as Online Analytical Processing (OLAP).

These two operations take place within the same domain and on the same data; however, they are different in nature. OLTP needs the client request to be served in a low response time and answered with the freshest data. In contrast, OLAP allows for a margin of low response time as well as less fresh data. Therefore, any cloud data storage system design has to adapt both scenarios to allow the system to function properly.

ES2 [4] is a cloud data storage system that supports both OLAP and OLTP within the same framework. The system architecture has a data access interface for upper application layer and it is responsible for receiving jobs from both OLAP and OLTP. Snapshot isolation is used for both kinds, which allows OLAP to be executed on potentially stale but consistent data while OLTP queries are being executed in parallel on a more recent snapshot.

5.4 Percolator

BigTables is a data storage built upon Google File System (GFS), and it is used mainly inside Google to store massive amount of data required by Google Apps, such as Web indexing, Map Reduce, and GoogleMaps. BigTable is not a relational database; rather, it maps two arbitrary string values, row key and column key, in addition to a time-stamp into an associated array of bytes.

To provide an efficient yet accurate approach to improve indexing updates, Peng and Dabek [11] propose Percolator, a system that is able to update Web indices in incremental processing without the need to retrieve the entire web documents. This approach allows maintaining a large repository of documents while at the same time processing the indexing update whenever an associated Web document is crawled.

The implementation of Percolator is built on top of BigTable and provides basically two additional features: multi-row transaction, which adds the transactional feature to BigTable, and the observer framework, which initiates the transaction for indexing update once page indexing changes.

6 Conclusion

This report presents several related topics in the area of distributed systems: dynamic system reconfiguration, caching, and cloud data storage. We discuss the latest research on configuring distributed systems to maintain system scalability in order to achieve higher performance and to guarantee system availability in the presence of failure. The dynamic reconfiguration aims to distribute resources among different applications sharing the same physical machine; or, on the other hand, it tries to determine the number of machines to be assigned to each application in a server cluster within a dynamic workload environment. In caching, the report presents several caching protocols and possible architectures for multi-tier systems. Finally, several database

platforms for the cloud were discussed which focus on providing the ACID property and exploring alternative processing approaches for large data analytical transactions.

References

1. A. Abouzeid, K. BajdaPawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. In *Proceedings of International Conference on Very Large Databases (VLDB)*, pages 922–933, 2009.
2. M. Attar and M. zsu. Alternative architectures and protocols for providing strong consistency in dynamic web applications. *World Wide Web*, 9(3):215–251, 1990.
3. G. Soundararajan C. Amza and E. Cecchet. Transparent caching with strong consistency in dynamic content web sites. In *Proceedings of ACM International Conference on Supercomputing (ICS)*, pages 264–273, 2005.
4. Y. Cao, C. Chen, F. Guo, D. Jiang, Y. Lin, B. C. Ooi, H. Tam Vo, S. Wu, and Q. Xu. Es2: A cloud data storage system for supporting both oltp and olap. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 291–302, 2011.
5. G. Casale, A. Kalbas, D. Krishnamurthy, and J. Rolia. Automatic stress testing of multitier systems by dynamic bottleneck switch generation. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware)*, pages 394–413, 2009.
6. J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content web servers. In *Proceedings of IEEE/ACM International Conference on Autonomic Computing (ICAC)*, pages 231–242, 2006.
7. D. Dash, V. Kantere, and A. Ailamaki. An economic model for selftuned cloud caching. In *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, pages 1687–1693, 2009.
8. S. Elnikety, S. Dropsho, E. Cecchet, and W. Zwaenepoel. Predicting replicated database scalability from standalone database profiling. In *Proceedings of European Conference on Computer Systems (EuroSys)*, pages 303–316, 2009.
9. S. Elnikety, S. Dropsho, and W. Zwaenepoel. Tashkent+: Memoryaware load balancing and update filtering in replicated databases. In *Proceedings of European Conference on Computer Systems (EuroSys)*, pages 399–412, 2007.
10. P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of European Conference on Computer Systems (EuroSys)*, pages 289–302, 2007.
11. D. Peng and F. Dabek. Large-scale incremental processing using distributed transactions and notifications. In *Proceedings of USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 1–15, 2010.
12. F. Perez-Sorrosal, M. Patio-Martnez, R. Jimnez-Peris, and B. Kemme. Consistent and scalable cache replication for multi-tier j2ee applications. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware)*, pages 328–347, 2007.
13. J. Philipp, N. De Palma, F. Boyer, and O. Gruber. Self adapting service level in java enterprise edition. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware)*, pages 143–162, 2009.
14. J. Rao, X. Bu, C. Xu, L. Wang, and G. Yin. Vconf: a reinforcement learning approach to virtual machines auto configuration. In *Proceedings of IEEE/ACM International Conference on Autonomic Computing (ICAC)*, 2009.
15. A. Soror, U. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *Proceedings of ACM International Conference on Management of Data (SIGMOD)*, pages 953–966, 2008.
16. G. Soundararajan, J. Chen, M. Sharaf, and C. Amza. Dynamic partitioning of the cache hierarchy in shared data centers. In *Proceedings of Very Large Database Endowment (PVLDB)*, pages 635–646, 2008.
17. G. Tesauro. Online resource allocation using decompositional reinforcement learning. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 886–891, 2005.
18. G. Tesauro, N. Jong, R. Das, and M. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. In *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER)*, pages 287–299, 2007.
19. Z. Wei, G. Pierre, and C. Chi. Cloudtps: Scalable transactions for web applications in the cloud. Technical report, Vrije Universiteit, Amsterdam, The Netherlands, 2010.