Cloud-Aware Processing of MapReduce-Based OLAP Applications

Hyuck Han\textsuperscript{1} Young Choon Lee\textsuperscript{2} Seungmi Choi\textsuperscript{1} Heon Y. Yeom\textsuperscript{1}

\textsuperscript{1} School of Computer Science and Engineering, Seoul National University, Seoul, 151-742, Korea.
Email: \{hhuyuck, smchoi, yeon\}@dcslab.snu.ac.kr

\textsuperscript{2} Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW 2006, Australia.
Email: yclee@it.usyd.edu.au, albert.zomaya@sydney.edu.au

Abstract

As the volume of data to be processed in a timely manner soars, the scale of computing and storage systems has much trouble keeping up with such a rate of explosive data growth. A hybrid cloud combining two or more clouds is emerging as an appealing alternative to expand local/private systems. However, the effective use of such an expanded cloud system is limited primarily by low network bandwidth and high latency between clouds (i.e., large intercloud data transmission overheads) when applications/services span across clouds, and they deal with large data in particular. Online analytical processing (OLAP) applications are a typical class of data-intensive application. These applications process multi-dimensional analytical queries dealing with ‘big data’ (or data warehouses). In this paper, we address the effective processing of MapReduce-based OLAP applications in a hybrid-cloud environment, and present a (hybrid) cloud-aware OLAP system incorporating data filtering techniques. Our system filters out unnecessary data for intercloud transmission with the ultimate goal of optimizing the performance to cost ratio, or cost efficiency. Based on experimental results obtained using two large-scale data analysis benchmarks, our system demonstrates its efficacy in improving the cost efficiency with the reduction in intercloud network traffic from 76%-99%.

Keywords: Cloud Computing; Hybrid Cloud; MapReduce; On-Line Analytical Processing (OLAP); Cost Efficiency

1 Introduction

Cloud computing with the support of virtualization technologies and utility computing (or pay-as-you-go) has emerged as a cost-effective solution for many computing tasks including large-scale data processing (Amazon Web Services 2012). For example, CycleComputing’s Amazon EC2 (Elastic Compute Cloud) powered 51,132 core high-performance cluster performed massive molecular modeling simulations—21 million chemical compounds—that used the equivalent of 12.5 CPU years for less than $4,900 an hour (CycleComputing 2012). In essence a cloud is classified as private or public based primarily on the availability to the public. A hybrid cloud can be formed as a mixture of two or more clouds of these two categories. Services offered in clouds can be classified into three types: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). This study takes a particular interest in IaaS clouds. An IaaS cloud provisions virtual resources such as computing nodes, storage, and networks, e.g., Amazon EC2 and S3 (Simple Storage Service).

In the recent past, we have witnessed dramatic increases in the volume of data literally in every area—business, science, and daily life to name a few. Today, some claim that data (more specifically, data-intensive science) are the fourth paradigm in scientific research after experimentation/observation, theory, and computational simulation (Hey et al. 2009). The storage and processing of such an overwhelming amount of data is a challenging task in the current computing environments. What’s more, the timeliness of data processing is the key to judicious decision making, particularly in business.

The MapReduce framework (Dean & Ghemawat 2008) proposed by Google is a parallel-programming model primarily for (large) data processing specifically designed with the consideration of large-scale distributed computing systems, such as clusters and data centers. A MapReduce application (or job) typically consists of large numbers of map and reduce tasks. Each map/reduce task deals with a chunk of data independently, and thus tasks in the job can be readily parallelizable and effectively processed in a large-scale computing environment like a cloud. Recently, MapReduce have been used in not only the analysis of a homogeneous data set (e.g., log processing) but also that of a heterogeneous data set such as data warehouses and data marts. Particularly in businesses, online analytical processing (OLAP) applications can take great advantage of the MapReduce framework because these applications process multi-dimensional analytical queries dealing with data warehouses (or ‘big data’).

It is often the case that the capacity of a single computer system (e.g., a private cloud) cannot keep up particularly with the growth rate of data volume to be processed by large-scale data processing applications, such as OLAP applications. Besides, private clouds occasionally get overloaded due to workload surges. The expansion with a public cloud (or hybrid cloud) is an appealing alternative to complement private clouds (Ceceti et al. 2010, Buyya et al. 2010, Johnston 2009).
Although adequate computational resources are available for MapReduce-based OLAP jobs in hybrid cloud environments, performance is not as high as expected due primarily to low bandwidth and high latency between private and public clouds. This intercloud (or cloud to cloud) transmission imposes an economic burden on users since network usage is also charged in public clouds. Thus, in hybrid cloud environments data movement is a crucial factor not only for performance, but also for cost.

In this paper, we address the problem of processing MapReduce-based OLAP applications in hybrid cloud environments. The work in this paper is designed to mitigate the performance and cost issues of MapReduce jobs in hybrid cloud environments by reducing the amount of intercloud data transfer using data filtering techniques. We exploit two types of filters—static and dynamic—deployed on the distributed file system; and they are evaluated with Hadoop. Our experimental results with data analysis workloads in Amazon EC2 demonstrate performance improvement and cost-cutting effects. Specifically, our OLAP system reduces network traffic as much as 99% (and at least 76%), and improves application performance (reduction in processing time) by 13-71%; together, 84% of total costs when processing without our system (default) is reduced.

The main contributions of this paper are as follows:

- We identify the impact of intercloud data transfer overheads on the performance of MapReduce-based OLAP applications.
- We develop a cloud-aware OLAP system based on the MapReduce framework for hybrid clouds.
- We demonstrate the effective usage of data filtering techniques to reduce intercloud data transmission overheads.
- Our system using two large-scale data analysis benchmarks has been evaluated in terms of both performance and cost efficiency.

The remainder of the paper is organized as follows. Section 2 presents background and related work. Section 3 describes the problem we address in this paper. Section 4 details the design and implementation of our system with description of data filtering technique incorporated. Section 5 evaluates the efficacy of our system in terms of performance (running time) and cost efficiency. Then, Section 6 concludes the paper.

2 Background and Related Work

In this section, we begin by describing the MapReduce framework and OLAP applications in the context of Hadoop (Apache 2012a), the open-source counterpart of MapReduce. We then discuss the deployment of OLAP applications in a hybrid cloud and issues related to such deployment.

2.1 MapReduce and OLAP

MapReduce is derived from functional programming concepts and is composed of two basic computation units/functions: Map and Reduce.

- Map takes an input and produces a set of intermediate key/value pairs. The MapReduce runtime classifies all intermediate values according to the same intermediate key $k$ and passes them to the Reduce function.
- Reduce receives an intermediate key $k$ and a set of values associated with the key. Reduce merges the values to form a set of new values. Typically, one output value is produced per one Reduce invocation.

Figure 1 shows the data flow in the Map/Reduce phases with IO for reading and writing data indicated by arrows. These IO activities are handled by the Hadoop Distributed File System (HDFS) resides in the private cloud. The existing Hadoop filtering is performed in worker nodes on which Map/Reduce functions execute instead of HDFS nodes; and some of these worker nodes are in the public cloud in our hybrid cloud model. Thus, for these worker nodes, the existing filtering approach has no effect on inter-cloud data traffic.

The MapReduce programming model has many advantages, such as high throughput/performance, use of commodity clusters, and fault tolerance. MapReduce is used in not only index construction for search engines (Dean & Ghemawat 2008) but also data analysis of both homogeneous and heterogeneous sets (Yang et al. 2007, Apache 2012c). Data join processing, which is very important for complex analysis in data warehouses, is addressed in (Yang et al. 2007) and (Pike et al. 2005) using MapReduce. Recently, Hadoop-based implementations, such as Hive (Apache 2012b) and CloudBase (Business.com 2012), have been developed for data warehouse workloads. In (Stonebraker et al. 2010), the authors compare MapReduce and parallel DBMS in various viewpoints such as performance and system management. Based on their conclusion, parallel DBMSs are suitable for efficient querying of large structured data, whereas MapReduce has advantages at complex analytics and extract-transform-load (ETL) tasks. This means that MapReduce can be useful for OLAP processing in large data warehouses. For example, Facebook has implemented a large data warehouse system using MapReduce instead of DBMSs (Monash 2009).

A data warehouse is an online repository for data from operational systems of an enterpise (W.H. Inmon 1996). A data warehouse is usually maintained using a star schema that is composed of a single fact table and any number of dimension tables. A fact table contains atomic data or records for business areas such as sales and production. Dimension tables have a large number of attributes that describes records of the fact table. Figure 2 shows an example of a star schema derived from the Star Schema benchmark database (O’Neil et al. 2007). The fact table is the LINEORDER table, and the dimension tables are CUSTOMER, SUPPLIER, PART, and DATE tables. The LINEORDER table has several foreign keys such as CUSTKEY, PARTKEY, SUPPKEY, and COMMITDATE to refer to each dimension table. Generally, queries in data warehouses are complex and ad hoc. The star-join query, in which the fact table...
is joined with one or more dimension tables, is one of the well-known queries in OLAP. For another example, TPC-H (Transaction Processing Performance Council 2012) provides a set of ad hoc queries for decision support systems that primarily use data in a data warehouse. In MapReduce, all data in data warehouses are stored as a form of a chunk in distributed file systems such as HDFS and Google File System (GFS) (Ghemawat et al. 2003). In this paper, we focus on MapReduce-based OLAP applications in data warehouses.

2.2 Hybrid Cloud Deployment

In (Vaquero et al. 2008), authors define a cloud as a large pool of easily usable and accessible virtualized resources. Clouds can also be regarded as data center hardware and software being served, and these resources are exposed in a pay-as-you-go style to enable public utility computing (Arnbrecht et al. 2010, Paul 2008). Cloud computing users take advantage of this pool of resources by paying only for the resources as their needs grow or shrink. The elasticity and scalability are the key characteristics of cloud computing platforms. Scalability is one rising issue in large-scale cloud deployments particularly with multi/many core machines (Boyd-Wickizer et al. 2010, Song et al. 2011). Authors in (Vaquero et al. 2011) extensively surveyed cloud scalability issues and classified them in three different levels.

As the scale of data constantly increases timely data processing and analysis is of great practical importance in business activities and scientific research communities such as HPC (Humphrey 2011, Zhai et al. 2011). If the data size is very large to the extent that a single cloud system cannot process the data, the cloud system would not be able to satisfy further requests from clients. In such a situation, the hybrid cloud plays a crucial role in resolving the scarcity of resources. If a private cloud in an enterprise becomes saturated, the enterprise may provision resources by renting from public cloud service providers. Thus, large-scale data processing in data warehouses will be an important target of hybrid clouds. However, current MapReduce frameworks do not consider hybrid clouds yet, and MapReduce-based OLAP applications on hybrid clouds experience low performance. This expansion of running OLAP application to multiple clouds offers a type of scalability solution, i.e., the platform level scalability as classified by authors in (Vaquero et al. 2011).

3 Problem Statement

In this section, we use an illustration to state the problem of processing MapReduce-based OLAP applications in hybrid clouds. Supposing there was a private cloud running a data warehouse, it may occasionally require more computing capacity beyond its own. If new physical machines are added to the private cloud, the heavy burden of both capital and operating costs would be inevitable. Thus, renting additional computing units from public clouds is naturally a cost-effective alternative. In this study, we do not consider a situation that the private cloud needs a public cloud for storage (e.g., security issues of internal data), and data are not partitioned or distributed over between private and public clouds.

In hybrid cloud environments, networking between clouds has low bandwidth and high latency since their communication essentially relies on the Internet. This is a major limiting factor to the adoption and prevalence of hybrid clouds, particularly for the MapReduce framework for which the data movement is frequent and large. Figure 3 demonstrates the effect of MapReduce applications when data are transferred from one cloud to another. While map tasks $M_1$ and $M_2$ receive their inputs from the data warehouse of their local cloud, map tasks $M_3$ and $M_4$ in a public cloud get their inputs from the private cloud, i.e., intercloud data transmission. Although all map tasks start almost at the same time, the map tasks in the public cloud take a longer time than the other two map tasks in the private cloud to execute due to the intercloud data transmission. Besides, each reduce task performs only after it gets corresponding output from every map task. Thus, no reduce tasks can start until the map tasks in the public cloud are finished, even if the other map tasks are completed. As a result, the map tasks in the public cloud are bottlenecks of total MapReduce processing, and the MapReduce program is slowed down.
4 Cloud-aware OLAP

In this section, our system design and implementation are articulated with the description of data filtering techniques incorporated. Then, cost efficiency metrics associated with our approach are presented.

4.1 Intercloud Transmission Reduction using Data Filtering

We propose a simple filtering technique that avoids transmission of unnecessary data particularly to public cloud nodes in which Map and Reduce tasks execute. And, we modify HDFS to be aware of data (e.g., record layout).

In our system, filters are configured in MapReduce programs by users (MapReduce Program Generation in Figure 4), and our system uses filters for better performance (MapReduce Task Running in Figure 4). When users write a MapReduce program, they add static/dynamic filters into the MapReduce program. Then, our system uses filters that users specified. When map processes start, they request data for their jobs with filter information to file systems (3 in Figure 4). File systems send filtered data to map processes on the public cloud (4 in Figure 4).

Currently, our system supports two types of filter: static (shown as 1.a in Figure 4) and dynamic (shown as 1.b in Figure 4) filters. A static filter, such as a relational algebraic operator, is recognized when a MapReduce task starts. Figure 5 indicates a sample database and its example query. A static filter uses a fixed constraint, such as $DT1.PK_1 = a_0$ and $DT2.PK_2 = b_0$ in Figure 5. This information is included as a job configuration parameter in the MapReduce Program (shown as 1.a Figure 4). It will be used to filter data from dimension tables (DT1 and DT2) if map tasks are placed outside the private cloud.

On the other hand, a dynamic filter, such as a bloom filter (Bloom 1970), can be used after filter construction is performed. For example, a user writes an efficient MapReduce-based join program by using a bloom filter. To this end, the user typically writes the program considering a filter construction phase (Business.com 2012, Han et al. 2011). During this phase, records of DT1 and DT2 are processed to produce bloom filters for all join keys ($PK_1$ and $PK_2$) and bloom filters are stored to the distributed file system. In the next step, records of the fact table (FT) and its corresponding dimension table are processed to perform the join processing. In this phase, the distributed file system first checks whether each foreign key ($FK_1$ and $FK_2$) of each record in the fact table, which usually is the largest table in the data warehouse, is contained in bloom filters from the previous phase or not. Then, if a record passes through the check, it is sent to map processes when they are far from the data warehouse cloud. Otherwise, it is dropped in the file system node. Due to the nature of the dynamic filters, the address of each bloom filter is saved as a configuration in MapReduce Program, (1.b) in Figure 4. It is noted that all bloom filters can be used in map processes if users do not use our system, and this technique can improve the performance of star-join queries in a single cloud (Han et al. 2011).

4.2 System Implementation

In this section, we give implementation details of our system including filter configuration as part of the OLAP MapReduce configuration and the actual filtering operation of data.

4.2.1 Filter Configuration

The filter information is stored as part of a job configuration in Hadoop. The information includes a type of filter (dynamic or static, or both), the use of filter (true, false), required arguments (e.g., addresses of bloom filters), and the location of the file system (whether inside or outside the private cloud).

Table 1: Filter property.

<table>
<thead>
<tr>
<th>Filter type</th>
<th>Dynamic filter</th>
<th>Static filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter type</td>
<td>(e.g., bloom filter)</td>
<td>(e.g., operators such as &lt; and &gt;)</td>
</tr>
<tr>
<td>filter address</td>
<td>number of filters</td>
<td>number of columns</td>
</tr>
<tr>
<td>line or only column</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 shows that each setting of filters needs a different configuration. The dynamic filter needs a type of filter, notifies its own type, and specifies addresses of bloom filters. The static filter also demands a filter type, the number of columns for the table of data, the number of filters, and a method to eliminate a line or column.

4.2.2 Operation on HDFS

In this section, we propose a transmission algorithm that applies a filter at each time after reading data. Since we consider the LineRecordReader class as
a default reader\(^{1}\), the first line and last line in the block (or chunk) may be overlapped with the previous or next chunk and may not be complete. We exclude these lines from the filter operation. The rest of the lines are passed on to the filtering step. If those lines are necessary data, they are added to the output; otherwise, they are discarded. When the output size is larger than or equal to the predefined packet size (e.g., 64KB), the file system nodes transfer the packet to map tasks in the computational nodes. The filtering operation (Algorithm 1) repeats until the end of the block is reached.

Algorithm 1 HDFS Filter.

\[
\text{while not the end of block do} \\
\quad \text{read data 64KB at a time} \\
\quad \text{if data are the first or last line of block then} \\
\quad \quad \text{continue} \\
\quad \text{end if} \\
\quad \text{apply filter to data} \\
\quad \text{concatenate filtered data to output} \\
\quad \text{if output is larger than packet size (64KB) then} \\
\quad \quad \text{transmit output} \\
\quad \text{end if} \\
\text{end while} \\
\text{transmit remaining output}
\]

We modify client and server code of original HDFS to realize filters. In the BlockSender class of the server side, data is recognized as a set of tuples, and is sent to the client side applying filters. The DFSClient.BlockReader class of the client side receives filtered data. For prototype implementation, we do not use existing checksum data stored in HDFS node but BlockSender re-computes checksum values when it sends filtered data. Additionally, we insert a one-byte value (boolean value) that indicates the last packet for the chunk. Since existing communication protocols use only the length of static data, the value is used to mark the completion of data transmission. To configure filters in MapReduce programs, we introduce new key/value pairs to JobConf objects, and the key/values pairs describe filter information shown in Table 1.

4.3 Cost Efficiency

The performance to cost ratio (cost efficiency) from the user’s perspective is an important metric when considering the use of public clouds in particular. In this section we characterize cost efficiency of running OLAP applications in hybrid clouds. We explicitly take into account intercloud network traffic and usage of resources (or instances in Amazon EC2) in our cost efficiency metrics.

The perfect linearity or even decent direct proportionality between the number of public resources rented and performance improvement is only in the ideal scenario (theory). Although this non-linearity is not only present with the use of public clouds, the cost related to public cloud usage makes such non-linearity more serious. This non-linearity is sourced from two main factors, particularly in our study with OLAP applications: (i) data transmission overhead between clouds and (ii) ‘hourly-base’ rate for public cloud resource rental. In the following, we devise a cost efficiency metric considering both factors.

For a given MapReduce-based OLAP job with \( M \) map tasks, we estimate the completion time of map phase \( T_m \) as follows:

\[
T_m = (T_i \cdot M)/(N_p + sd \cdot N_b)
\]

where \( T_i \) is the average execution time of map tasks, \( N_p \) and \( N_b \) are the numbers of resources in the private cloud and the public cloud, respectively, and \( sd \) is a slowdown rate. \( sd \) in our study is primarily estimated based on the first non-linearity factor described above, i.e., intercloud data transmission. Clearly, the performance of public cloud resources is affected/decreased by the amount of data to be transferred. We do not consider the reduce phase because it is performed in a private (local) cloud.

The expense of renting public cloud resources \( C_b \) is calculated by the product of unit resource cost \( C_i \), total map phase time (hour) \( T_m \), and the number of rented resources \( N_b \).

\[
C_b = C_i \cdot [T_m] \cdot N_b
\]

The cost efficiency of running a MapReduce job in a hybrid cloud is the reduced time per unit price, i.e., the performance improvement to the public cloud cost ratio. It is determined by \( T_m \) and \( T_d \) where \( T_d \) is the map phase time without the support of the public cloud. More formally,

\[
CE = (T_d - T_m)/C_b.
\]

5 Experimental Evaluation

In this section we detail experiments to evaluate our system and present results. Specifically, the performance improvement capability of our system is verified with experiments in the real cloud setup using Amazon EC2. Then, we show cost saving implications and discuss how to optimize cost efficiency.

5.1 System Performance

In this section, we show that our system reduces the amount of data transmission and this reduction leads to performance improvement effectively.

5.1.1 Experimental Environment

For our experiment, we rented resources from Amazon EC2 (Standard Small, m1.small) in two different available regions: US East and West zones as shown in Figure 6. We used four instances in each region as computing nodes, and four more instances were added to the East zone as storage nodes. Map tasks in the US East zone receive their input data from the same zone (as in the private cloud). However, map tasks in the US West zone receive their input from the US East zone through an inter-zone (intercloud) network.

Figure 6: Experimental environment.
Our system was evaluated with well-known, large-scale data analysis benchmarks:

- **TPC-H** - a business-oriented decision support benchmark, which simulates an online production database environment; we used a scale factor of 20 (i.e., a data set of 20GB) in this study.

- **SSBM (Star Schema Benchmark)** - a benchmark for data warehousing applications; we used a scale factor of 40, which also gives us a database size of about 20GB. SSBM has four query groups, and each group has a couple of queries with different selectivity of the `LineOrder` table (fact table). We used the MapReduce-based SSBM benchmark suite (Han et al. 2011).

These benchmarks provide dedicated data generator programs (`db_gen`). Each `db_gen` program produces data files (.tbl files). For example, data for the `lineitem` table is stored in the `lineitem.tbl` file. A tuple of a table corresponds to a line of a data file, and each column in a tuple is separated by a special identifier (`|`). In our experiments, we uploaded all data files to HDFS or filter-enabled HDFS and Map/Reduce programs process data from distributed storage line by line. We ran each test case five times and measured the average running time and average traffic volume. The data are transferred between clouds in two different ways:

- **Default** - default Hadoop transmission
- **Filter_hdfs** - filtering data in storage nodes, where the data are filtered before the transmission of the data.

We report results of only dynamic filters (e.g., bloom filter) for this study since static filters do not show significant performance improvement (less than 10%). Because OLAP applications as our target applications have many complex join procedures, our dynamic filters can improve query performance and reduce network usage significantly. It is noted that all results include costs of additional filter construction phases (computation, network, and storage costs).

### 5.1.2 Results

Results are presented in two aspects: traffic volume and running time. Clearly, these two performance metrics are negatively correlated.

Figure 7 shows the efficiency of our system using `Filter_hdfs` for processing the TPC-H benchmark. Figure 7(a) also shows that performance using the default transmission—with explicit intercloud data filtering—is improved to a certain degree with the use of public cloud resources; however, this degree of performance improvement is not quite align with extra costs related to public cloud usage. Our system leads to further performance improvement through filtering-enabled HDFS. That is, `Filter_hdfs` reduces data transmission by 76-99% compared with the default hadoop transmission (Figure 7(b)), and this leads to performance improvement of 13-56% (Figure 7(a)).

Similar results were obtained from experiments with processing SSBM queries (Figure 8). From Figure 8(b), we can see that our system significantly reduces intercloud data transmission for star join queries (i.e., 95-99% reduction). This leads to superior performance of the `Filter_hdfs` transmission by 49.5-70.5% in terms of running time (Figure 8(a)).

More concrete data on performance improvement our system delivered in experiments are shown in Table 2.

![Figure 7: TPC-H results.](image)

![Figure 8: SSBM results.](image)
5.2 Cost Efficiency

This section begins by demonstrating actual cost savings obtained from our system and discusses cost efficiency implications. All costs are calculated based on the actual rates of Amazon EC2.\(^2\)

5.2.1 Cost Savings

We summarize total running times and intercloud traffic volumes of TPC-H and SSBM—with and without (default) using our system—in Table 2. Corresponding cost savings sourced from those reductions in Table 2 are plotted in Figure 9.\(^3\) Red bars indicate cost savings from reduction in intercloud network traffic while blue bars show cost savings from improved performance (fewer instance rental hours). Specifically, cost savings from intercloud traffic reduction are $4.8 and $8.7 for TPC-H and SSBM, respectively. Such cost savings can afford additional 56 and 102 default compute instances (m1.small) for 1 hour; and this implies further performance improvements ‘recursively’. Cost savings in instance rental from improved performance are $0.34 and $1.7 for TPC-H and SSBM, respectively. Overall, 84% of costs were reduced using our system, i.e., 96% in data transfer costs and 40% in instance rental.

<table>
<thead>
<tr>
<th>Table 2: Total running time &amp; intercloud traffic volume.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total running time (hour) Filter_hdfs default reduction</td>
</tr>
<tr>
<td>TPC-H</td>
</tr>
<tr>
<td>SSBM</td>
</tr>
<tr>
<td>Total intercloud traffic (GB) Filter_hdfs default reduction</td>
</tr>
<tr>
<td>TPC-H</td>
</tr>
<tr>
<td>SSBM</td>
</tr>
</tbody>
</table>

Figure 9: Cost savings.

5.2.2 Optimization of Cost Efficiency

As the usage of public cloud resources is charged based on the number of whole hours, the cost efficiency of cloud deployment is largely dependent on the compactness of tasks with a given number of resources. We have verified this cost efficiency characteristic using the example below. Given a MapReduce-based OLAP application, we consider that \( M = 3000 \) map tasks, \( T_i = 55 \) sec, \( C_i = $0.2, sd = 0.45, \) and \( N_p = 4. \)

![Cost Savings](image)

Figure 10 shows the relationship between cost efficiency and public cloud cost with respect to different volumes of public resource rental. In this experiment, we have identified eight ‘cost efficiency’ points for the number of resources to be rented (resource count or \( N_b \)), i.e., they are good candidates for running the application in terms of cost efficiency. These candidates are 4, 6, 9, 12, 17, 26, 42, and 95 of resources in this experiment as indicated by vertical lines in Figure 10; they are the points where running times are a multiple of whole (or very close to whole) hours, i.e., 7.90, 6.85, 5.69, 4.88, 3.93, 2.91, 1.99, and 0.98. Specifically, each of these points is the case that an additional public resource contributes to the reduction of running time by one hour. Thus, public cloud cost drops as shown in Figure 10 and Table 3 despite the increase in the number of public cloud resources. Table 3 highlights this hourly rate originated pattern exhibited between 25 and 44 of public resources in our experiment. Clearly, the global maximum and its corresponding number of resources are the best interest of the user. In this particular experiment, renting 42 instances is the best choice in terms of cost efficiency (performance improvement per dollar). However, one may select another point (a local maximum) due to budget or time constraints. The cost efficiency characteristic presented in this section can greatly facilitate the design of scheduling and resource allocation policies for the user. Note that since good candidates for the number of public resources to be rented appear around multiples of hours, the search for the global maximum terminates at around one hour of running time.

6 Conclusion

To date, a majority of use cases of hybrid cloud deployments lie in with computationally intensive applications. Yet, applications and services deployed in cloud environments are increasingly data intensive. Cloud sourcing—delegating the entire IT solution to public clouds—might be an alternative; however, it is often not quite possible due to various reasons including security. Thus, reducing the amount of intercloud data transmission is of great practical importance in hybrid cloud deployments. In this paper, we have studied on the intercloud data transmission of OLAP applications, and presented a cloud-aware OLAP system using data filtering techniques. We have shown that our system is capable of reducing intercloud data transmission significantly; that is, experimental re-

\(^2\)Resource rental: $0.085/hr for Linux/Unix m1.small, and data transfer in/out: $0.1 and $0.15 per GB, respectively.

\(^3\)Total costs in our experiments do not include costs for the private cloud since the management or pricing policy of the private cloud may vary.
suits verified this claim with improvements in both running time and cost efficiency. We also have explicitly taken into account the current practice of (public) cloud service pricing, and devised cost efficiency metrics to discuss about judicious resource rental decisions.

Acknowledgements

This work was supported by Mid-career Researcher Program through NRF grant funded by the MEST (No. 2010-0014387). The ICT at Seoul National University provided research facilities for this study.

References

Humphrey, M. (2011), Cloud, hpc, or hybrid: A case study involving satellite image processing, in ‘Cloud Futures 2011’.
Johnston, S. (2009), ‘Intercloud is a global cloud of clouds’.
W.H. Inmon (1996), Building the Data Warehouse, J. Wiley & Sons, Inc.