Collaborative Multi-dimensional Dataset Processing with Distributed Cache Infrastructure in the Cloud

Youngmoon Eom, Jonghwan Moon, Jinwoong Kim, and Beomseok Nam

School of Electrical and Computer Engineering,
Ulsan National Institute of Science and Technology,
Ulsan, Republic of Korea, 689-798
{*youngmoon01,choco1916,jwkim,bsnam*}*@unist.ac.kr*

***Abstract*— As modern large scale systems are built with a large number of independent small servers, it is becoming more important to orchestrate and leverage a large number of distributed buffer cache memory seamlessly. Several previous studies showed that with large scale distributed caching facilities, traditional resource scheduling policies often fail to exhibit high cache hit ratio and to achieve good system load balance. A scheduling policy that solely considers system load results in low cache hit ratio, and a scheduling policy that puts more emphasis on cache hit ratio than load balance suffers from system load imbalance. To maximize the overall system throughput, distributed caching facilities should balance the workloads and also leverage cached data at the same time. In this work, we present a distributed job processing framework that yields high cache hit ratio while achieving good system load balance, the two of which are most critical performance factors to improve overall system throughput and job response time. Our framework is a component-based distributed data analysis framework that supports geographically distributed multiple job schedulers. The job scheduler in our framework employs a distributed job scheduling policy - *DEMA* that considers both cache hit ratio and system load. In this paper, we show collaborative task scheduling can even further improve the performance by increasing the overall cache hit ratio while achieving load balance. Our experiments show that the proposed job scheduling policies outperform legacy load-based job scheduling policy in terms of job response time, load balancing, and cache hit ratio.**

I. Introduction

In distributed job processing systems, the size of distributed buffer caching facilities scale with the numer of distributed servers. Leveraging the large distributed caches plays an important role in order to improve the overall system throughput as many systems are being built by connecting small heterogeneous machines. In distributed environment orchestrating a large number of distributed caches to improve overall cache hit ratio is not an easy problem. The traditional job scheduling policies such as Round-Robin and load-monitoring are not sophisticated enough to consider cached results in distributed servers but they only consider load balance [1].

Since job scheduling policies such as Round-Robin do not consider cache hit ratio, the utilization of distributed caches will be low since cached data items in remote servers will not be considered. On the contrary, scheduling policies that make scheduling decisions solely based on cache hit ratio will not balance system load and the system will suffer from load imbalance. For example, if a certain server has very hot cached items, that single server will be flooded with a large number of tasks while the other servers are all idle. In order to achieve load balancing as well as to exploit cached results, it is required to employ more intelligent job scheduling policies than the traditional Round-Robin and load-monitoring scheduling policies.

In this paper, we study a distributed job scheduling policy - *DEMA* that makes job scheduling decisions based on *spatial clustering* so that similar jobs get clustered together in order to improve cache hit ratio and the number of jobs in each cluster becomes balanced. Our proposed spatial clustering algorithm is different from the well known spatial clustering algorithms used in machine learning in a sense that the job scheduling decisions must be made at run time, i.e., the overhead of run-time scheduling algorithms should be minimal. DEMA scheduling policies take into account the dynamic contents of distributed caching infrastructure in order to exploit cached results and employ statistical prediction methods to balance load across distributed servers.

In order to manage the distributed caching infrastructure in a seamless fashion and to facilitate collaborative scheduling over geographically dispersed data repositories and application servers, our distributed job processing framework employs multiple front-end servers that periodically synchronize workload statistics to make better job scheduling decisions. A salient feature of the cloud computing is that it enables collaborative research and facilitates access to remote computing resources. In the context of distributed cloud environment, multiple front-end schedulers are likely to reduce job response time as they accelerate scheduling decisions. Our extensive experimental studies show our proposed job scheduling policy and multiple job schedulers significantly improve the job processing throughput and outperforms legacy load based scheduling policy by a large margin.

The rest of the paper is organized as follows: Section II describes other resesarch works related to cache-aware job scheduling policies in distributed sytems. In section III, we describe overall architecture of our distributed data processing framework. In section IV, we discuss DEMA scheduling policy and how the scheduling policy works with multiple geographically distributed schedulers. In Section VI we eval-

uate our scheduling policy with data migration improvement. In section VII, we conclude.

II. Related Work

Load-balancing problems have been extensively investigated in many different fields. Godfrey et. al [2] proposed an algorithm for load balancing in heterogeneous and dynamic peer-to-peer systems. Catalyurek et. al [3] investigated how to dynamically restore balance in parallel scientific computing applications where the computational structure of the applications change over time. Vydyanathan et. al [4] proposed scheduling algorithms that determine what tasks should be run concurrently and how many processors should be allocated to each task. Zhang et. al [5] and Wolf et al. [6] proposed scheduling policies that dynamically distribute incoming requests for clustered web servers. WRR (Weighted Round Robin) [7] is a commonly used, simple but enhanced load balancing scheduling policy which assigns a weight to each queue (server) according to the current status of its load, and serves each queue in proportion to the weights. However, none of these scheduling policies were designed to take into account a distributed cache infrastructure, but only consider the heterogeneity of user requests and the dynamic system load.

LARD (Locality-Aware Request Distribution) [8], [9] is a locality-aware scheduling policy designed to serve web server clusters, and considers the cache contents of back-end servers. The LARD scheduling policy causes identical user requests to be handled by the same server unless that server is heavily loaded. If a server is too heavily loaded, subsequent user requests will be serviced by another idle server in order to improve load balance. The underlying idea is to improve overall system throughput by processing jobs directly rather than waiting in a busy server for long time even if that server has a cached response. LARD shares the goal of improving both load balance and cache hit ratio with our scheduling policies, but LARD transfers workload only when a specific server is too heavily loaded while our scheduling policies actively predict future workload balance and take actions beforehand to achieve better load balancing.

In relational database systems and high performance scientific data processing middleware systems, exploiting similarity of concurrent queries has been studied extensively. That work has shown that heuristic approaches can help reuse previously computed results from cache and generate good scheduling plans, resulting in improved system throughput as well as reducing query response times [10], [11]. Zhang et al. [12] evaluated the benefits of reusing cached results in a distributed cache framework in a Grid computing environment. In that simulation study, it was shown that high cache hit rates do not always yield high system throughput due to load imbalance problems. We solve the problem with scheduling policies that consider both cache hit rates and load balancing.

In order to support data-intensive scientific applications, a large number of distributed query processing middleware systems have been developed including MOCHA [13], DataCutter [14], Polar* [15], ADR [16], and Active Proxy-G [17]. Active Proxy-G is a component-based distributed query processing grid middleware that employs user-defined operators that application developers can implement. Active Proxy-G employs meta-data directory services that monitor performance of back-end application servers and a distributed cache indexes. Using the collected performance metrics and the distributed cache indexes, the front-end scheduler determines where to assign incoming queries considering how to maximize reuse of cached objects [11]. The index-based scheduling policies cause higher communication overhead on the front-end scheduler, and the cache index may not predict contents of the cache accurately if there are a large number of queries waiting to execute in the back-end application servers.

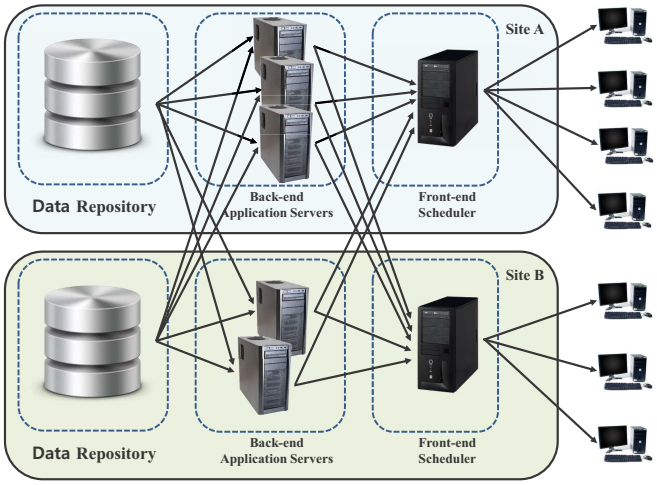


Fig. 1. *Architecture of Distributed Query Processing Framework*

III. Distributed and Parallel Query Processing Framework

Figure 1 shows a simplified architecture of our distributed and parallel job processing middleware for data analysis applications. The goal of this architecture is to build an efficient and scalable job processing framework that achieves load balancing, parallel processing, and high cache hit ratio with a distributed semantic caching infrastructure. The front-end server interacts with clients for receiving jobs and returning job results. The back-end application servers typically run on cluster nodes, shared memory machines, or distributed shared memory machines with attached large-scale storage devices or networked databases.

The front-end server determines which back-end application server processes which incoming job. In many distributed job processing middlewares including Active Proxy-G [18], the front-end server employs a directory service that collects performance metrics from application servers, and balances the system load across the back-end application servers by assigning jobs to the least busy application server. More naive scheduling policy is Round-Robin, where one of the

application servers is chosen one after another in a predefined order.

Over the past years, we have developed a scientific data analysis processing framework - *UniDQP* to efficiently execute multiple scientific workloads for data analysis applications on distributed environment [19], [11], [18]. UniDQP is a component-based distributed job processing system that is currently under development. UniDQP provides well-defined programming model so that scientific data analysis application developers can implement the interface of user-defined operators to process scientific data analysis on top of UniDQP framework. Many scientific data analysis applications are different in terms of the raw dataset type and the task type. However, they have common features in the overall job processing flow and many scientific datasets are represented in a multi-dimensional space. Image information, geographical data are examples of this kind of dataset which can be represented in a multi-dimensional space and have spatial locality.

One of the most common access patterns into such scientific datasets is multidimensional range query. Application-specific user-defined operators retrieve raw datasets from the storage systems and process incoming jobs to generate the corresponding outputs. UniDQP provides a multidimensional indexing module, which helps back-end application servers identify which raw datasets should be accessed for a given range query or which cached results should be reused. Application developers must implement an interface class - *UserQuery*, which specifies how to access raw datasets, how to generate outputs, and how to map an input data to the multidimensional problem space. UniDQP simultaneously handles incoming jobs in a multi-threaded environment. The front-end server manages a large number of connections with clients and back-end application servers, and performs job scheduling that employs dynamic scheduling algorithms that we will describe shortly.

IV. Predicting Cached Data Items in Distributed Semantic Caches

In this section, we describe our job scheduling policy which employs a statistical prediction method called *Exponential Moving Average* (EMA) for assigning an incoming job to an application server that is most likely to have cached data item required by the job. In addition, our scheduling algorithm balances the workloads among the servers regardless of the distribution of cached data items.

A. Background: Exponential Moving Average (EMA)

Exponential moving average (EMA) is a well-known statistical method to obtain long-term trends and smooth out short-term fluctuations; applications are found in finance such as predicting stock prices and trading volumes [20].

The EMA computes a weighted average of all historical data by assigning exponentially more weight to recent data.

Let p_t be the cached object at time $t > 0$ and EMA_t be the computed average at time t after adding p_t into the cache. Given the *smoothing factor* $\alpha \in (0, 1)$ and the previous

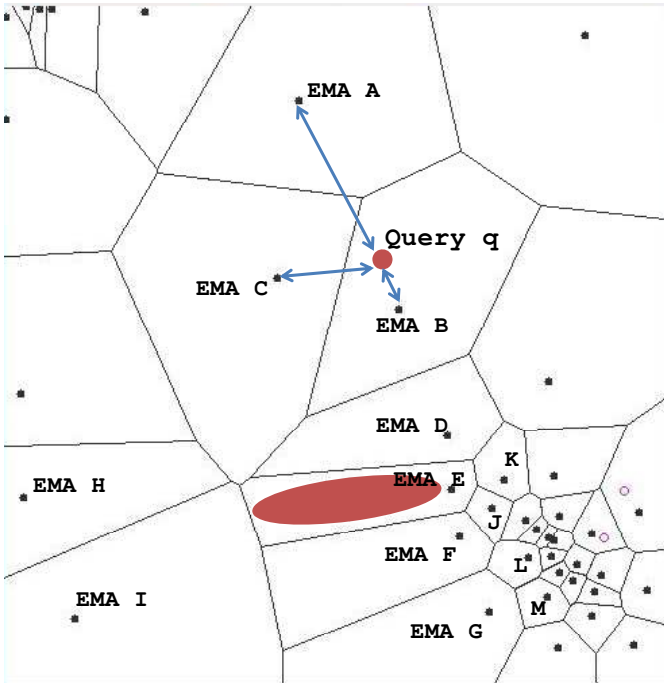


Fig. 2. *DEMA scheduler calculates the Euclidean distance between EMA points and an input data needed by an incoming job, and assigns the job to the server B whose EMA point is closest.*

average EMA_{t-1}, the next average EMA_t can be defined incrementally by

$$EMA_t = \alpha \cdot p_t + (1 - \alpha) \cdot EMA_{t-1} \quad (1)$$

and it can be expanded as

$$EMA_t = \alpha p_t + \alpha(1 - \alpha)p_{t-1} + \alpha(1 - \alpha)^2 p_{t-2} + \cdots \quad (2)$$

Since our framework is supposed to run long enough to amortize the initial EMA error, we simply chose $k = 1$, i.e., $EMA_1 = p_1$ where p_1 is the first observed data. Equation 1 is used for the following EMAs. Note that EMA value can be a multi-dimensional element depending on the data domain of the application.

The smoothing factor α determines the degree of weighing decay towards the past. For example, α close to 1 drastically decreases the weight on the past data (short-tailed) and α close to 0 gradually decreases (long-tailed).

B. Distributed Exponential Moving Average (DEMA)

The *Distributed Exponential Moving Average (DEMA)* scheduling policy [11] employs as many EMA values as the back-end application servers to determine which server is to be assigned each job and how the assignment adapts to the dynamic change of the distribution of input datasets. DEMA scheduling algorithm works in three steps.

a) 1. Initialization.: For simplicity, let a unit circle represent the *problem space* where the value starts from zero and increases in a clockwise direction up to one which meets at zero. We assume that each input data is mapped to a point in multi-dimensional problem space using its center. Suppose

there are n back-end application servers that process user jobs and one scheduling server that assigns each incoming job to an application server, as shown in Figure 1. In the beginning, the scheduling server creates n EMA variables EMA_1, $EMA_2, \ldots,$ EMA_n, with EMA_i for the ith application server, and then assigns the first n input data to EMA_1, $EMA_2, \ldots,$ EMA_n, respectively.

b) 2. Assignment.: Let q denote the center point of an input data requested by an incoming job, ranging from 0 to 1, and let q also represent the job itself. Upon receiving a job q, the scheduling server finds the EMA point EMA_{i^*} that is closest to q (in terms of their Euclidean distance), and then assigns q to server-i^*. For example, in Figure 2 an input data q is closest to EMA point B, hence an incoming job that needs the input data q is assigned to server B.

c) 3. Update.: Once q is assigned to server-i^*, the scheduling server updates the EMA point for the server by

$$EMA_{i^*} = \alpha q + (1 - \alpha) EMA_{i^*}. \tag{3}$$

d) Complexity.: In step 2, the scheduling server needs to find the closest EMA point to a given input data. This algorithmic problem, known as Nearest Neighbor Search, can be solved in $O(\log n)$ using space partitioning data structure such as *k-d tree* or *R-tree*. [1]

C. Cache hit of DEMA

Note that an application server keeps only certain number of recent input data in its cache; for example, it may employ Least Recently Used (LRU) policy. Let L denote the current number of cached input data in the cache. Ideally, it is desired that the EMA point reflects only those in the cache. However, EMA reflects all the past data including those that were expunged from the cache, but with less weight for older ones. Therefore, we may want to choose the decay factor α such that the weight sum of the expunged input data is managed below a threshold. Formally, given a threshold $\epsilon \in (0, 1)$, we want

$$(1 - \alpha)^L < \epsilon. \tag{4}$$

Since L depends on the size of cached objects, we use the average number of input data in the cache instead, denoted by L^*. Therefore, we can choose the smallest α that meets

$$\alpha > 1 - \epsilon^{1/L^*}. \tag{5}$$

In this way, the EMA value of an application server reflects what are current in the cache, and assigning incoming jobs that are close to the EMA, only jobs that need similar input data (i.e., close to each other in problem space) are assigned to the application server. This clustering effect promotes the cache hit ratio in the application server.

[1] In our implementation, however, we performed a linear search that takes $O(n)$ assuming that n is small enough, for example, up to 40.

D. Load Balancing of DEMA

Another important feature of DEMA scheduling algorithm is *load balancing*; each job can be assigned to one of the application servers with equal probability, leveling the workloads of the application servers. To that end, DEMA scheduling algorithm gradually moves the EMA points so that they follow the distribution of requested input data. If a certain region of problem space becomes hot, the EMA points move to the hot spot by EMA equation (3). If input data requests are uniformly distributed, then DEMA algorithm tries to keep the sizes of Voronoi regions as similar as possible. DEMA algorithm achieves this goal by evenly distributing EMA points throughout the problem space.

Let us assume that the problem space is a 2-dimensional plane and the input data requests follow a uniform distribution. Figure 2 is a snapshot of the Voronoi diagram where a dot represents an EMA value of an application server (e.g., EMA B is the EMA value of server B), and a line segment represents the set of Euclidean middle points between two EMA points. In DEMA scheduling, an incoming job that reads input data which falls in a Voronoi cell of an EMA point makes the EMA point slide toward the input data point, where the amount of the movement is affected by the smoothing factor α (the more α, the more shift). In Figure 2, the job that needs the input data q will be forwarded to server B since the EMA point of server B is closer to the input data point than any other EMA points. It is called the *Voronoi assignment model* where we assign every multidimensional point to the nearest cell point. The job assignment regions induced from the DEMA assignment form a *Voronoi diagram* [21].

In this case, EMA B moves a little toward q, moving all the border lines of the cell a little toward the same direction. An important observation is that each EMA has tendency to move toward the center of its cell. For example, EMA E in the figure is located at the right corner of the cell. Thus more jobs will arrive to the left of the EMA in the cell than to the right. Therefore, the EMA E is more likely to slide to the left rather than to the right. This movement will also move the border to the left, making the Voronoi region of EMA I smaller and EMA J and K larger. This property hints at the mechanism of load balancing; DEMA scheduling algorithm has a tendency to make a larger cell (EMA I) smaller and a smaller cell (EMA K) larger.

V. Collaborative Scheduling in the Cloud

The Cloud environment facilitates collcaborative work and allows many clients to execute jobs over geographically distributed computing resources. In such geographically distributed systems, job submission systems had better be near clients to hide network latency when they interact with the system. Thus our distributed query processing framework can be configured to work with multiple schedulers as shown in Figure 1.

With multiple schedulers, a client will submit a job to its closest scheduler, but the scheduler may forward the job to a back-end server which is geographically far remote from

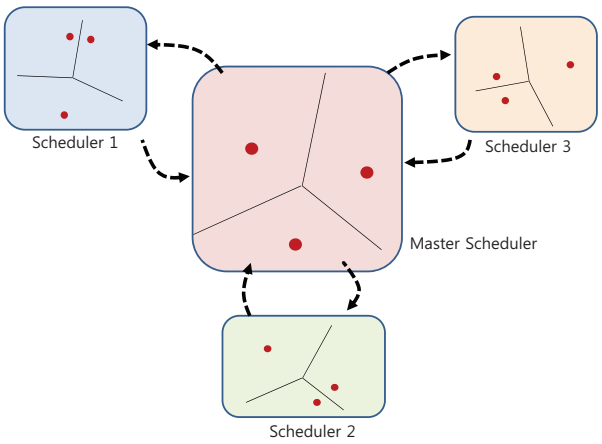


Fig. 3. *Multiple schedulers share EMA points by synchronizing their own EMA points periodically.*

the scheduler. The schduler needs to consider all available statistics about the system such as cached data objects, current system load, network latency between organizations, storage location, etc.

As described in section IV, DEMA scheduling policy makes scheduling decisions based on geometric features of past jobs. One assumption with DEMA scheduling policy is that one schduler has seen all the past jobs, and it predicts which back-end server has what cached data items and how many past jobs were forwarded to which back-end server. With multiple schedulers, this assumption does not hold true any more. One scheduler may receive very different types of jobs from another scheduler. In such cases, EMA points in each schduler can be very far from each other, and it hurts the spatio-temporal locality of the jobs running in each back-end server, and results in poor load balance with low cache hit ratio.

In order to solve the problem, this work compares simple solutions that periodically exchange the EMA points between schedulers, compute the weighted average EMA points of back-end servers, and share the updated EMA points for subsequent incoming jobs as shown in Figure 3. The synchronization process does not have to block processing incoming jobs since unsynchronized EMA points may slow down the job execution time slightly but it might be better to immediately schedule incoming jobs rather than waiting for the synchronization.

A. Synchronization of EMA points

One of the simple synchronization methods that we study is *averaging method*. In averaging method, each scheduler independently updates its own EMA points, and a master scheduler periodically collects EMA points of other schedulers, computes the average EMA points, and broadcasts the updated EMA points to all other schedulers. The frequency of computing average EMA points should be determined so that network overhead is minimal. But if the frequency is chosen to be too low, EMA points in each scheduler will diverge and the overall system throughput will hurt.

A problem of the averaging method is when the number of scheduled jobs in each schduler varies. For example, if a schduler $S1$ received very few number of jobs while another schduler $S2$ scheduled a large number of recent jobs, the EMA points in $S1$ do not reflect the recent cached data objects in back-end servers. Therefore our improved synchronization method - *weighted averaging method* gives more weight to EMA points of a server which received more queries than other schedulers. The weight is determined by how many jobs were forwarded to each server, i.e., if a back-end server did not receive any job from a scheduler, the weight of EMA point of the back-end server is 0. The weighted average EMA point of each back-end server is calculated as the following equation:

$$EMA_{i^*} = \frac{\sum_{s=1}^{n} N_i[s] \times EMA_{i^*}[s]}{\sum_{s=1}^{n} N_i[s]} \quad (6)$$

where $N_i[s]$ is the number of jobs assigned to server s by scheduler i.

In both synchronization methods, the frequency of synchronization is a critical performance factor that affects the overall system throughput. As we compute the average EMA points more frequently, each scheduler will have more accurate information about cached data objects in back-end servers. If the synchronization interval is very large, each back-end server's buffer cache will be filled with data objects that do not preserve spatio-temporal locality as in round-robin policy.

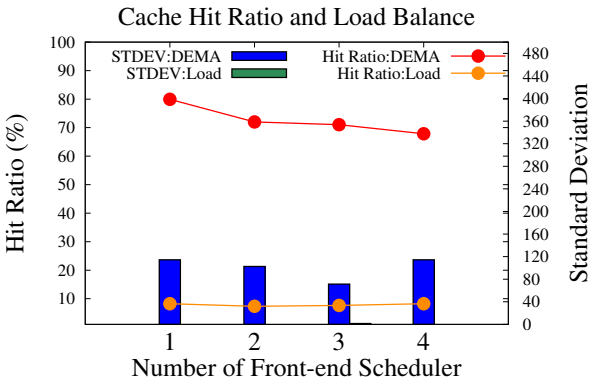
VI. Experiments

Performance improvements from planning and scheduling are highly dependent on the nature of applications, the system characteristics, and also the characteristics of the experimental workload. In order to show the magnitude of improvements that can be expected from employing DEMA-based scheduling policies, we perform studies using various input data distributions and realistic job workloads.

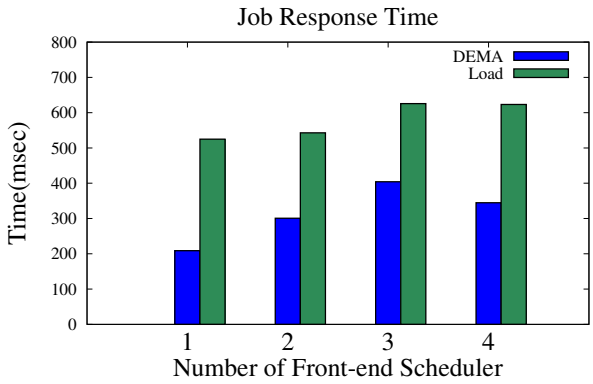
In order to show that the DEMA-based scheduling policies perform well with various input data distributions, we synthetically generated workloads using uniform, normal, and Zipf's probability distributions with a spatial data generator implemented by Yannis Theodoridis, which is available at [22]. In order to measure how periodic synchronization of EMA points adapt to various input data requests, we submit jobs that access different input datasets in various distributions to each scheduler. For the experiments, each scheduler schedules 10,000 jobs that need input dataset tagged with 2-dimensional spatial coordinates, (e.g, latitude and longitude).

A. Experiments Result

In our experiments we employ four homogeneous clusters, each of which consists of a single front-end scheduler and 5 back-end application servers. Note that in our distributed system configuration, a front-end scheduler can assign jobs

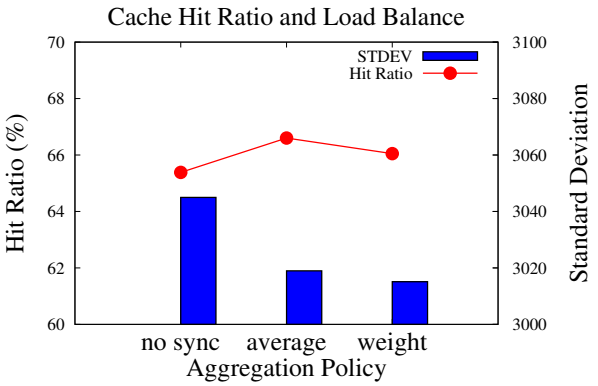


(a) Cache Hit Ratio and Load Balance

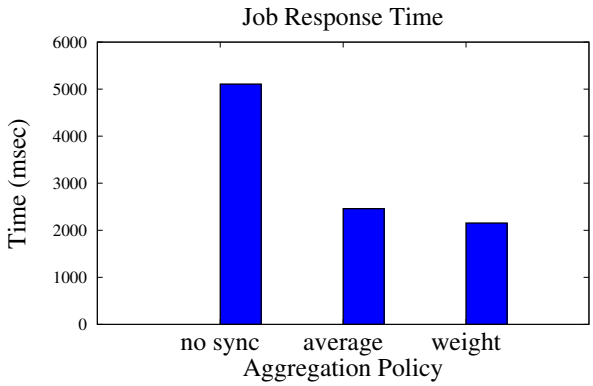


(b) Job Response Time

Fig. 4. *Performance Comparison Varying Number of Schedulers*



(a) Cache Hit Ratio and Load Balance



(b) Average Query Response Time

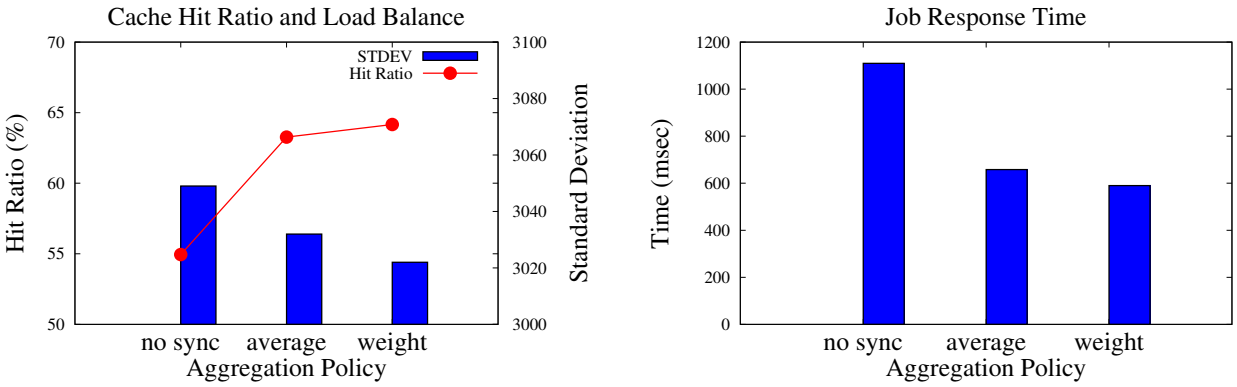
Fig. 5. *The Performance Comparison of Aggregation Policy(5 msec of inter-arrival time)*

to any back-end server in remote sites. If the cluster servers are not homogenous, schedulers should apply weight to the distance between EMA points and input data points so that more powerful servers receive more jobs. As for the network latency between remote sites, our experiments do not consider it but it can be also taken into consideration by applying appropriate weight to the distance between EMA points and input data points.

For the experiments shown in Figure 4, we varied the number of schedulers while the number of back-end application servers is fixed to 20. To measure performance, we use cache hit ratio, load balance, and job response time as performance metrics. Job response time is measured from the moment a job is submitted to a scheduler until the job is finished. Hence the job response time includes the waiting time in a job queue and it depends on the load balance and cache hit ratio. When there is only a single scheduler, the cache hit ratio of DEMA scheduling policy is 80% while the hit ratio of load-based scheduling policy is just 8.25%. As load based scheduling policy solely considers the system load, its standard deviation of scheduled jobs in back-end servers is almost 0 while the standard deviation of DEMA scheduling policy is higher than 50. However, due to very high cache hit ratio, the average job response time of DEMA scheduling policy is about 1.55∼2.51 times faster than that of load-based scheduling policy. Note that as the number of schedulers increases, the cache hit ratio of DEMA scheduling policy decreases slightly because EMA points in each scheduler are likely to diverge with more schedulers. As a result, the job response time of DEMA scheduling policy also increases slightly as the number of scheduler increases, but still it shows significantly lower job response time.

Figure 5 and 6 show the job response time, cache hit ratio, and load balancing in terms of the standard deviation of number of processed jobs in each server. Label $average$ and $weight$ shows the performance of the averaging method and the weighted averaging method respectively that we described in section V. $nosync$ shows the performance of DEMA scheduling policy that does not synchronize the EMA points of multiple schedulers.

As the ratio of job inter-arrival time and synchronization frequency can affect the effectiveness of synchronizing EMA points, we varied the job inter-arrival time and measured the performance of DEMA scheduling policy. In Figure 5, we fixed the synchronization frequency to 400 msec, and the job inter-arrival time was 5 msec on average.



(a) Cache Hit Ratio and Load Balance

(b) Average Query Response Time

Fig. 6. *The Performance Comparison of Aggregation Policy(10 msec of inter-arrival time)*

As expected, the naive $nosync$ scheduling policy shows the worst performance because of relatively lower cache hit ratio and higher standard deviation due to the divergence of EMA points across multiple schedulers. The averaging method exhibits the highest cache hit ratio, but weighted averaging method is better in terms of load balance. As a result, weighted averaging method shows the fastest query response time.

In the experiments shown in Figure 6, the job inter-arrival time was increased to 10 msec while the synchronization frequency was 400 msec. For the same reason, the naive $nosync$ scheduling shows the lowest cache hit ratio and the worst load balance resulting in the higest average job response time. The weighted averaging method is again showing best performance in terms of both cache hit ratio and load balance. In both graphs, Figure 5 and Figure 6, the averaging method and weighted averaging method perform significantly better than $nosync$ scheduling.

For the experiments shown in Figure 7 and Figure 8 we varied the synchronization interval and measured the performance of the weighted average synchronization methods. As we decrease the synchronization interval, the communication overhead between schedulers increase but the schedulers can make scheduling decisions with more accurate prediction of cached data objects.

Fig 7 shows the cache hit ratio, standard deviation of number of processed jobs in each server, and the average job response time with varying the synchronization interval. The job inter-arrival time is fixed to 5 msec. When the synchronization frequency is too high (synchronization per 4 jobs), system load balance is slightly worse than lower synchronization frequency, but its cache hit ratio is highest among others. But as the synchronization interval increases, the cache hit ratio decreases and the load balance also becomes poor. In terms of the job response time, we observe the fastest job response time when synchronization interval is 50. With too high synchronization interval, the response time significantly increases because outdated EMA points hurt cache hit ratio and load balance, but too frequent synchronization also does not help improving job response time due to synchronization overhead.

Fig 8 shows the performance with varying the synchronization interval when the average job response time is 10 msec. Since the job inter-arrival time is doubled compared to the experiments shown in Fig 7, it shows the best performance when the synchronization interval is 100. Similarly, load balancing is not significantly affected by the synchronization interval but the cache hit ratio decreases as the synchronization interval increases.

VII. Conclusion

In this paper, we discuss a distributed job scheduling policy - *DEMA* and how the scheduling policy can be used when multiple schedulers collaborate managing distributed computing resources in the cloud environment.

Our experiments show the distributed job scheduling policy and its simple extension for multiple schedulers outperform legacy load based scheduling policy in terms of cache hit ratio and load balancing.

References

[1] B. Nam, D. Hwang, J. Kim, and M. Shin, "High-throughput query scheduling with spatial clustering based on distributed exponential moving average," *Special issue on data intensive eScience, Distributed and Parallel Databases*, vol. 30, no. 5–6, pp. 401–414, 2012.

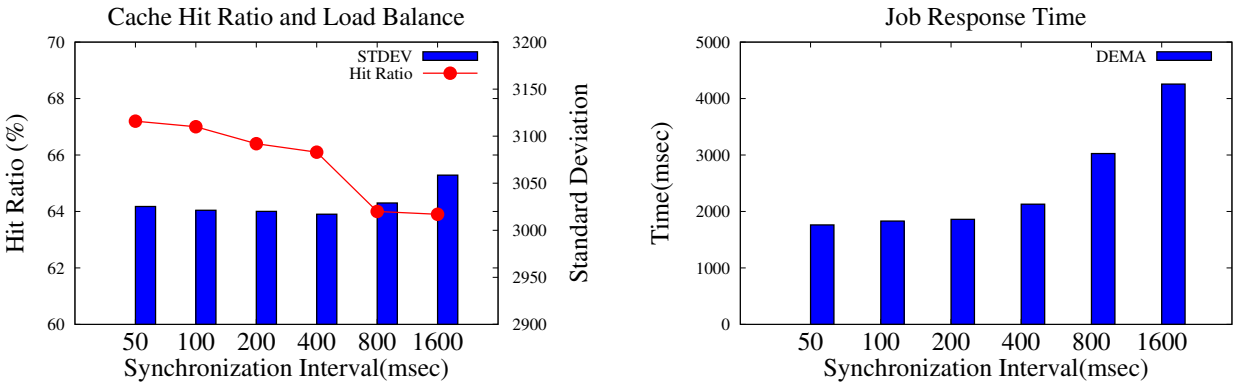
[2] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured p2p systems," in *Proceedings of INFOCOM 2004*, 2004.

[3] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdag, R. T. Heaphy, and L. A. Riesen, "A repartitioning hypergraph model for dynamic load balancing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 8, pp. 711–724, 2009.

[4] N. Vydyanathan, S. Krishnamoorthy, G. Sabin, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz, "An integrated approach to locality-conscious processor allocation and scheduling of mixed-parallel applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, pp. 3319–3332, 2009.

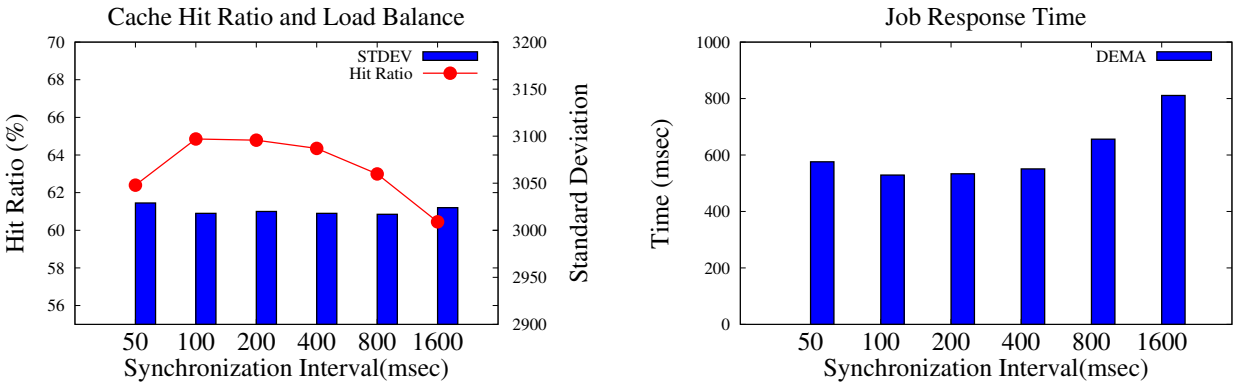
[5] Q. Zhang, A. Riska, W. Sun, E. Smirni, and G. Ciardo, "Workload-aware load balancing for clustered web servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 3, pp. 219–233, 2005.

[6] J. L. Wolf and P. S. Yu, "Load balancing for clustered web farms," *ACM SIGMETRICS Performance Evaluation Review*, vol. 28, no. 4, pp. 11–13, 2001.



(a) Cache Hit Ratio and Load Balance (b) Average Query Response Time

Fig. 7. *The Performance Effect of Synchronization Interval(5 msec of inter-arrival time)*



(a) Cache Hit Ratio and Load Balance (b) Average Query Response Time

Fig. 8. *The Performance Effect of Synchronization Interval(10 msec of inter-arrival time)*

[7] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose atm switch chip," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1265–1279, 1991.

[8] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, "Scalable content-aware request distribution in cluster-basednetwork servers," in *Proceedings of Usenix Annual Technical Conference*, 2000.

[9] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-aware request distribution in cluster-based network servers," in *Proceedings of ACM ASPLOS*, 1998.

[10] J.-S. Kim, H. Andrade, and A. Sussman, "Principles for designing data-/compute-intensive distributed applications and middleware systems for heterogeneous environments," *Journal of Parallel and Distributed Computing*, vol. 67, no. 7, pp. 755–771, 2007.

[11] B. Nam, M. Shin, H. Andrade, and A. Sussman, "Multiple query scheduling for distributed semantic caches," *Journal of Parallel and Distributed Computing*, vol. 70, no. 5, pp. 598–611, 2010.

[12] K. Zhang, H. Andrade, L. Raschid, and A. Sussman, "Query planning for the Grid: Adapting to dynamic resource availability," in *Proceedings of the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, Cardiff, UK, May 2005.

[13] M. Rodríguez-Martínez and N. Roussopoulos, "MOCHA: A self-extensible database middleware system for distributed data sources," in *Proceedings of 2000 ACM SIGMOD*.

[14] M. D. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz, "DataCutter: Middleware for filtering very large scientific datasets on archival storage systems," in *Proceedings of the Eighth Goddard Conference on Mass Storage Systems and Technologies/17th IEEE Symposium on Mass Storage Systems*. National Aeronautics and Space Administration, Mar. 2000, pp. 119–133, nASA/CP 2000-209888.

[15] J. Smith, S. Sampaio, P. Watson, and N. Paton, "The polar parallel object database server," *Distributed and Parallel Databases*, vol. 16, no. 3, pp. 275–319, 2004.

[16] T. Kurc, C. Chang, R. Ferreira, A. Sussman, and J. Saltz, "Querying very large multi-dimensional datasets in ADR," in *Proceedings of the ACM/IEEE SC1999 Conference*. ACM Press, Nov. 1999.

[17] H. Andrade, T. Kurc, A. Sussman, and J. Saltz, "Active Proxy-G: Optimizing the query execution process in the Grid," in *Proceedings of the ACM/IEEE SC2002 Conference*, Nov. 2002.

[18] H. Andrade, T. Kurc, A. Sussman, and J. Saltz, "Multiple query optimization for data analysis application clusters of SMPs," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*. IEEE Computer Society Press, May 2002.

[19] B. Nam, H. Andrade, and A. Sussman, "Multiple range query optimization with distributed cache indexing," in *Proceedings of the ACM/IEEE SC2006 Conference*, 2006.

[20] Y. lun Chou, *Statistical Analysis*. Holt International, 1975.

[21] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry, Algorithms and Applications*. Springer, 1998.

[22] Y. Theodoridis, "R-tree Portal," *http://www.rtreeportal.org*.