Distributed and Parallel Databases, Volume 30, Issue 5-6, pp 401-414, Springer, 2012 manuscript No.
(will be inserted by the editor)

High-Throughput Query Scheduling with Spatial Clustering based on Distributed Exponential Moving Average (EMA)

Beomseok Nam · Deukyeon Hwang ·
Jinwoong Kim · Minho Shin*

June 29, 2012

Abstract In distributed scientific query processing systems, leveraging distributed cached data is becoming more important. In such systems, a front-end query scheduler distributes queries among many application servers rather than processing queries in a few high-performance workstations. Although many query scheduling policies exist such as round-robin and load-monitoring, they are not sophisticated enough to exploit cached results as well as balance the workload. Efforts were made to improve the query processing performance using statistical methods such as *exponential moving average*. However, existing methods have limitations for certain query patterns: queries with hotspots, or dynamic query distributions. In this paper, we propose novel query scheduling policies that take into account both the contents of distributed caching infrastructure and the load balance among the servers. Our experiments show that the proposed query scheduling policies outperform existing policies by producing better query plans in terms of load balance and cache-hit ratio.

1 Introduction

In many scientific disciplines, computational scientists and engineers generate and analyze enormous amounts of datasets to better understand complex physical phenomena. Efficient storage, retrieval, and processing of such large

Beomseok Nam · Deukyeon Hwang · Jinwoong Kim are at Electrical and Computer Engineering, Ulsan National Inst. of Science and Technology, Ulsan, 689-798, Korea · Minho Shin (corresponding author) is at Dept. of Computer Engineering, Myongji University, Yongin, Gyonggido, Korea.

This research was supported by the 1.100027.01 Research Fund of the UNIST(Ulsan National Institute of Science and Technology) and 2.110147.01 National Research Foundation of Korea. · This work was also supported by 2011 Research Fund of Myongji University. This manuscript is for the purpose of self-archiving, and the final publication is available at link.springer.com.

scientific datasets are now major challenges that need to be addressed for scientific analysis applications. Distributed and parallel query processing systems have been used to solve large and complicated scientific problems as its parallelism enabled substantial gain in performance.

For maximum parallelism, *load balancing* must be achieved so that all users' tasks are partitioned and evenly distributed across parallel servers and keep the servers busy all of the time. However, load balancing in modern heterogeneous cluster systems is not an easy task; even if the number of tasks is well balanced, it does not imply that the system throughput is maximized. On the other hand, for computationally expensive queries, *cache-hit ratio* also plays an important role in reducing the query response time. However, traditional scheduling policies such as round-robin ignores the cache-hit ratio in distributed servers, and the data in the caches is rarely reused. Therefore, a query scheduling policy should take into account both load balancing and the reuse of cache contents. For example, if a query scheduler knows the workload and the cache content of each server, the scheduler can maximize the parallelism and the cache-hit ratio.

A challenge of a cache-aware scheduling policy is that the scheduling server needs to know the content of each cache, which causes significant communication overhead between the scheduler and the application servers. The scheduling algorithm also needs to be lightweight so that it doesn't cause bottleneck at the front-end scheduler.

Another difficulty with a cache-aware query scheduling policy is that high cache-hit ratio may not result in high system throughput because it may hurt load balance. Suppose a query scheduler knows the cache content of an application server and those contents are popular. So it forwards most subsequent queries to the application server to improve cache-hit ratio. With proper caching policy, those popular data will remain in the cache for an extended time period, making the response time for those queries small. However, with such popular queries (called *hotspot*), it is difficult to maintain the load balance among the servers, degrading query response time. For example, if the server with popular data in the cache is too much busier than other servers, the waiting time in the busy server can be higher than the processing time by an idle server without the data in the cache. In such a case, queries should be forwarded to an idle server. This can improve system throughput and reduce query response time even if it decreases cache-hit ratio.

In a prior work [9], a query scheduling policy was designed to achieve both load balance and cache reuse. The policy employs a statistical prediction method called Exponential Moving Average (EMA). However, the policy fails to perform well for certain cases; it fails to balance the load when the query distribution has hotspots, and it fails to adapt to query patterns when the query distribution changes frequently. In this paper, we propose two novel

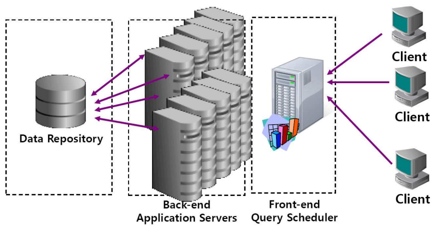


Fig. 1 *Architecture of Distributed Query Processing Framework*

scheduling policies that overcome the limitations of the prior work against hotspots and dynamic query distributions.

The first proposed policy takes into account the recent workload of the application servers while assigning similar queries to the same server for cache-hit ratio. The recent workload of each application server is estimated by the front-end scheduler without any communication overhead between the scheduler and the application servers. The second policy employs additional dynamics into the policy so that it adapts quickly to the change of the query distribution and balances the load in a global manner. More specifically, for each query we change the state of the most idle server as well as that of the application server that is assigned the query. Such change in the idle server not only draws the idle server out of its dormant state and assigns more queries to the server, but it can also make more servers to converge quickly to new hotspots.

We ran experiments with both synthetic and realistic 2D image query workloads. The experimental results show that the proposed scheduling policies quickly adapt to dynamic query distribution changes, and outperform the load-based scheduling policy by orders of magnitude for both static and dynamic query patterns.

The rest of the paper is organized as follows. In Section 2 we formulate the distributed query scheduling problem, and in Section 3 we describe the rational behind EMA-based scheduling techniques and propose new scheduling policies. Experimental settings and results are discussed in Section 4, and related work is given in Section 5. We conclude in Section 6.

2 Semantic Caching in Distributed Query Processing Framework

Figure 1 shows the architecture of distributed and parallel query processing middleware for scientific data analysis applications. This system aims for linear speed-up and scale-up via load balancing, processing queries in parallel, and increasing cache-hit ratio in distributed semantic caching infrastructure [1]. The front-end server, called *scheduler*, interacts with clients for receiving queries, and determines which back-end application servers will process incoming queries. The back-end application servers run application-specific user-defined operators that retrieve raw datasets from the storage systems and process incoming queries on cluster nodes.

Many scientific data analysis applications are different in terms of the raw dataset type and the query type. However, they have common features in the overall query processing flow and many scientific datasets are represented in a multi-dimensional space. One of the most common access types into scientific datasets is multidimensional range query. When a query is forwarded to an application server, the application server searches its semantic cache to find out if the given range query overlaps the range of cached objects to reuse. If no cached object is found in the semantic cache, it processes the query from scratch. The goal of such a system is to maximize the query processing throughput and minimize the query response time.

3 EMA-based Scheduling Policies

In this section, we describe how one can employ the statistical prediction method *Exponential Moving Average* (EMA) for clustering incoming queries around application servers, achieving both load balance and high cache-hit ratio. We first describe the basic algorithm, called DEMA (Distributed EMA) [9], then propose improved algorithms that overcome the drawbacks of DEMA.

3.1 Distributed EMA scheduling

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 [6]. In distributed query scheduling policies, the EMA computes a weighted average of all the past cached data by assigning exponentially more weights to recent data. As the cached data is represented by multi-dimensional coordinates, we used the multi-dimentional center point of the data to calculate the EMA value.

Let $p[t]$ be the multidimensional center point of cached object at time $t > 0$ and $EMA[t]$ be the EMA value at time t after adding $p[t]$ into the cache. Given the *smoothing factor* $\alpha \in (0,1)$ and the previous average $EMA[t-1]$, $EMA[t]$ can be calculated incrementally by

$$EMA[t] = \alpha \cdot p[t] + (1-\alpha) \cdot EMA[t-1] \tag{1}$$

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).

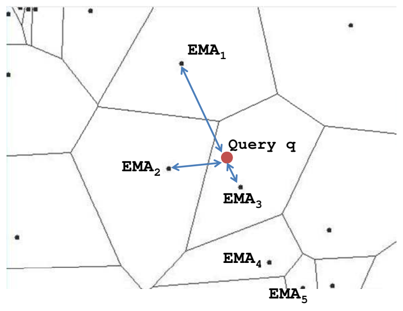


Fig. 2 *DEMA scheduler calculates the Euclidean distance between EMA points and an incoming query, and assigns the query to the server B whose EMA point is closest.*

In the distributed caching infrastructure, we employ as many EMA points as the application servers to determine which server is to be assigned each query and how the assignment adapts to the dynamic change of the query distribution.

The EMA-based scheduling algorithms work in three steps.

1. Initialization. Suppose there are n application servers (which processes queries) and one scheduler (which assigns each incoming query to an application server). For initialization, the scheduler chooses n EMA variables $EMA_1, EMA_2, \ldots, EMA_n$ uniformly at random, where EMA_i is associated with the ith application server (denoted by *server i*).

2. Assignment. For an incoming query q, the scheduler computes the center of the query, also denoted by q, then finds the closest EMA point EMA_{i^*} to q (in terms of Euclidean distance). Query q is then forwarded to server i^*.

3. EMA Update. Once q is assigned to server i^*, the scheduler updates EMA_{i^*} by

$$EMA_{i^*} = \alpha q + (1-\alpha) EMA_{i^*}. \tag{2}$$

The complexity of DEMA scheduling is $O(\log n)$ on average case where n is the number of application servers. This is so when we employ a nearest neighbor search algorithm [3] for finding the closest application server. The same is true in all query dimensions if the dimension is considered constant. For simplicity, however, we provide an $O(n)$ algorithm in a later section (see Algorithm 1), which performs linear search.

Figure 2 visualizes the EMA-based scheduling for two dimensional queries. In the figure, $EMA_1 \ldots EMA_5$ represent the EMA values of the five application servers. The boundaries represent the Voronoi diagram of the EMA values, where points on the boundary has the same distance to the closest EMA values. The scheduler assigns each query to the server whose Voronoi cell contains the center of the query, thus called *Voronoi assignment model* [4].

For example, the query q in the figure is assigned to server 3. By Equation 2, EMA_3 moves a little toward q, which also shifts the boundary of the cell toward the same direction. An observation is that each EMA has a tendency to position at the center of its cell. For example, EMA_4 in the figure is located at the lower right corner of the cell. Thus more queries will arrive to the upper left side of the EMA in the cell than to the lower right. Therefore, the EMA_4 is likely to slide to upper left. This movement will also move its borders to the upper left, making EMA_2 and EMA_3 smaller and EMA_5 larger. This property hints at the mechanism of DEMA for load balancing; DEMA scheduling algorithm has a tendency to make a larger cell (EMA_2) smaller and a smaller cell (EMA_5) larger.

However, the Voronoi assignment model doesn't balance server-load well in certain circumstances. If all the subsequent queries land on a single Voronoi cell, only the corresponding single application server will process all of them. If a popular query region suddenly moves to a distant location where only a few EMA points are located, DEMA scheduling policy may need substantial amount of time to adjust the distribution of EMA points accordingly. This delay prevents the balanced query distribution among the servers for an extended period of time.

The following describes how our new scheduling policies address these limitations of DEMA.

1. Instead of picking up a bisector line as the query boundary of two neighboring application servers, we choose the boundary according to the workload of two servers. The underlying idea of this approach is that a server that has more pending queries should process less number of subsequent queries. In this variant, the boundaries of EMA points are chosen considering the number of recently assigned queries.
2. Instead of updating one EMA point per query, we update multiple EMA points including the closest one. The underlying idea of this approach is to quickly adapt the EMA distribution to new query patterns.

3.2 Balanced Exponential Moving Average (BEMA)

The balanced DEMA policy (BEMA in short) employs a novel metric to represent the effective distance between the query and the application server. The *load-aware Euclidean distance* not only measures the Euclidean distance between the query point and the EMA point, it also takes into account the current load of the server. Formally, the load-aware Euclidean distance is defined as the Euclidean distance multiplied by the estimated workload of the server.

Algorithm 1 describes the BEMA algorithm in detail. In line 10, the algorithm uses the variable *WeightedDistance* for the load-aware Euclidean

Algorithm 1
BEMA Algorithm

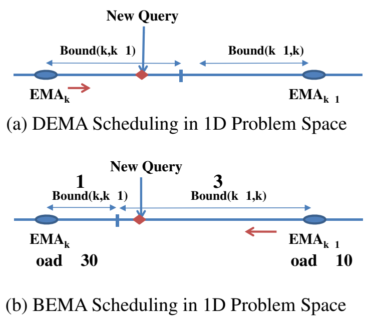
```
1: INPUT: a client query $Q$
2: $MinDistance \leftarrow MaxNum$
3: for $s = 1$ to $NumberOfApplicationServers$ do
4:     if EMA[s] is not initialized then
5:         forward query $Q$ to server $s$.
6:         $EMA[s] \leftarrow Q$;
7:         return;
8:     else
9:         $Load[s] \leftarrow GetCurrentLoadOfServer(s)$;
10:        $WeightedDistance \leftarrow EuclideanDistance(EMA[s], Q) \cdot Load[s]$;
11:        if $WeightedDistance < MinDistance$ then
12:            $SelectedServer \leftarrow s$
13:            $MinDistance \leftarrow WeightedDistance$
14:        end if
15:    end if
16: end for
17: forward query $Q$ to $SelectedServer$.
18: $EMA[SelectedServer] \leftarrow (\alpha \cdot Q) + (1 - \alpha) \cdot EMA[SelectedServer]$
```

distance between the server and the query. The algorithm obtains the server workload ($Load[s]$) using the function $GetCurrentLoadOfServer(\cdot)$. To measure the server workload, various metrics can be employed: size of query wait queue, disk read rate, and thread pool utilization, for instance. However, collection of such information requires additional communication with the servers. Instead of collecting performance metrics from the server, the scheduler estimates the server workload using *query assignment count*, which is the number of queries assigned to the server among the recent N queries. The query assignment count can be a good estimator for the wait-queue size when the queries are reasonably homogeneous. The benefit of this metric is that the scheduler can measure it without any communication with the application servers.

Clustering property of BEMA: As a query is assigned to an application server if it is close enough to the EMA point of the server, only similar queries (i.e., close to each other in the query space) are assigned to the same server. This clustering effect promotes the reuse of cache content, especially for range queries because similar range queries will overlap each other. This effect of BEMA promotes cache-hit ratio.

Load Balancing property of BEMA: Another important feature of BEMA scheduling algorithm is *load balancing*; each query can be assigned to one of the application servers with equal probability, leveling the workloads of the application servers.

The motivation of Balanced EMA (BEMA) algorithm is to assign more queries to less busy servers. The DEMA scheduling policy doesn't consider the workloads of the adjacent servers; it assigns the query to the server whose



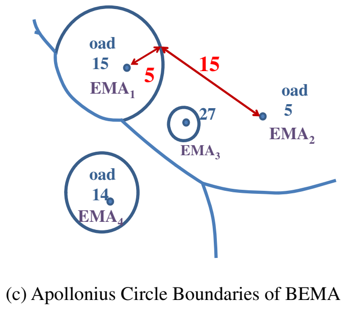


Fig. 3 *BEMA scheduler assigns the query to the server whose Apollonius circle encloses the query point.*

EMA point is the closest regardless of its current workload (see Figure 3-(a)). On the contrary, BEMA multiplies the query assignment count of the server, representing the workload of the server, to the Euclidean distance between the query and the EMA of the server. In Figure 3-(b), the load of server k is 3 times higher than that of server $k+1$. In the weighted distance, the server k becomes much farther from the query than server $k+1$, effectively shifting the decision boundary toward the server k. Thus the query q is assigned to server $k+1$, balancing the workload. In general, the BEMA scheduling policy chooses a boundary between EMA_i and EMA_j such that

$$Bound(i,j) : Bound(j,i) = Load(j) : Load(i)$$

where Bound(i,j) is the distance between EMA_i and the boundary to EMA_j, and $Load(i)$ is the query assignment count.

Ideally, we want to asign the same number of queries to each application server. DEMA scheduling policy fails to achieve load balancing if the query distribution has a hot spot, or if the query distribution changes frequently. In EMA-based scheduling policies, the probability of assigning a new query to a specific server equals to the probability of a new query belonging to the region of the server. The BEMA scheduling policy defines the regions by *multiplicatively weighted Voronoi diagram* whose boundaries are defined by Apollonius circles (Figure 3-(c)). The BEMA scheduling policy adjusts the Apollonius circular region size reciprocal to server load. For example, in Figure 3-(c) server 2 (EMA_2)'s load is about 3 times lower than server 1 (EMA_1). Hence the region of server 1 becomes much smaller than that of server 2.

Figure 4 illustrates how BEMA converges to load balance in the view of *min-max optimization*. For brevity, we assume linear query space and uniform query distribution. Suppose server 2 (with EMA value of $E2$) is a local maximum in terms of the workload; $Load(2) > Load(1)$ and $Load(2) > Load(3)$. By definition, boundary $m1$ is closer to $E2$ and $m2$ is closer to $E2$, making

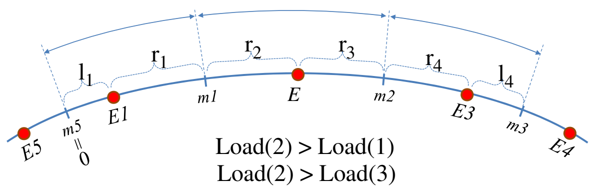


Fig. 4 *An example of a local maximum server.*

$r2 < r1$ and $r3 < r4$. Therefore, $r2 + r3 < r1 + r4$, i.e., the probability of a query being assigned to server 2 is less than being assigned to either server 1 or sever 3. This will decrease the load of server 2, minimizing the local maximum load.

3.3 Moving Multiple EMA for Better Load Balancing (MMEMA)

Suppose that queries suddenly start to land inside the boundary of a single application server's Voronoi cell. By DEMA, all subsequent queries will be assigned to the hotspot server. Although most queries will find the query results in the cache (i.e., cache-hit ratio close to 1), the server will be overwhelmed by the burden of processing all the queries while other servers remain idle. To address this problem, BEMA makes the hotspot Voronoi cell shrink very quickly and redistribute the load to adjacent servers. This property of BEMA also makes the EMA distribution quickly adapt to the change of query distribution.

Another approach to the problem is to move another server in addition to the closest one. Moving an additional EMA point toward the query helps to lessen the burden of the hotspot server; the '*supporting server*' quickly approaches to the hotspot (and it moves more quickly if it is farther from the hotspot, by EMA definition), then takes over part of the hotspot cell.

We can choose the supporting server at random, in a round-robin fashion, the farthest to the query, or we can move the EMA point of the most idle server. In this work, we chose the most idle server as the supporting server. The benefit of moving the most idle server is twofolds. Moving the most idle server toward the hotspot maximizes the effect of load balancing. Another benefit is that if a new hotspot suddenly starts to form in a non-hotspot area, both the closest EMA point and the most idle EMA point reaches to the new hotspot area, dividing the hotspot burden into two. This effect alleviates the hotspot problem much better than the case when only one EMA is responsible for the hotspot. Keeping track of the most idle server is also computationally lightweight if we keep the same load metric as in BEMA. The most idle server will not be changed unless it receives more queries while it travels toward the hotspot. During the move, the most idle server may arrive at a different hotspot and another server may become the most idle server.

4 Experiments

4.1 Experimental Setup

Performance improvements from query planning and scheduling highly depends on the nature of the application, the system characteristics, and the characteristics of the workload. In order to show the magnitude of improvements by the proposed scheduling policies, we evaluated various scheduling policies using different query distributions and realistic query workloads.

In order to evaluate the performance of the EMA-based scheduling policies under different query distributions, we generated synthetic queries that follow uniform, normal, and Zipf's probability distributions. In addition to the theoretic probability distributions, we employed a variation of the Customer Behavior Model Graph (CBMG) technique to generate more realistic query workloads [8]. In this model, the first query in a batch specifies a geographical region and the subsequent queries simulate user behavior based on the following operations: a *new point of interest*, *spatial movement (scanning)*, and *resolution increase or decrease*.

We have measured the performance of EMA-based scheduling policies on 41 Linux cluster machines, and also implemented a simulator to evaluate the performance of a larger number of machines. The machines are equipped with dual Quad-Core Xeon E5506 2.13 GHz CPUs and 12 GB of memory per node, and they are connected by Gigabit switched Ethernet. The datasets are stored in NFS mounted file system so that each server has access to the entire raw dataset. Two dimensional digital pathology image datasets were used to evaluate the query scheduling policies The images were partitioned into sub-images and the average query processing time to read a single sub-image is about 200 ms.

4.2 Experimental Results

In addition to EMA-based scheduling policies (i.e., DEMA, BEMA, and MMEMA), we also evaluated the Load-based scheduling policy for comparison. The Load-based scheduling policy (denoted by Load) assigns each query to the least busy server among all the servers to balance the load, but it does not consider cache content. As we will show, the EMA-based scheduling policies consistently outperform Load-based scheduling policy due to cache hit ratio.

Figure 5 and 6 show the change of the average query response time, cache-hit ratio, and load balance as the number of application servers increases. The

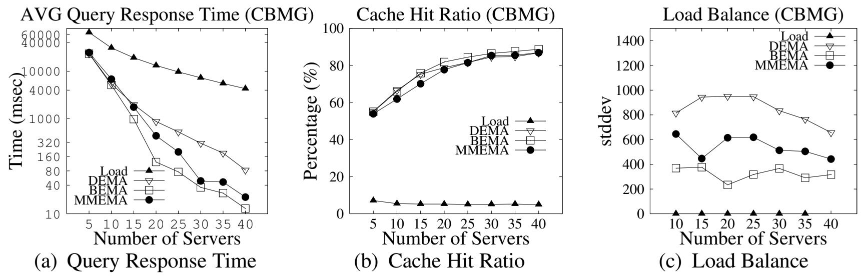


Fig. 5 *CBMG Query Distribution (# of App. Servers)*

query response time is defined as the time between when the scheduler receives the query and when the application server completes the query processing. This time includes the queueing delay in the application server as well as the actual processing time. Each application server has an LRU cache whose size can hold up to 100 query results.

In Figure 5(a), the query response time decreases as the number of servers increase. Additional servers provide more cache space to the distributed query system. Hence more intelligent scheduling algorithms that take into account the locality between queries better utilize the distributed large cache space. Figure 5(c) shows the change of load balance measured by the standard deviation of the number of assigned queries. BEMA shows superior load balancing behavior compared to other EMA-based scheduling policies with CBMG distribution. Although the BEMA scheduling policy shows similar cache-hit ratio with the DEMA scheduling policy (Figure 5(b)), BEMA shows the smallest query response time with CBMG distribution because it outperforms DEMA in terms of load balancing. Due to limited space, we omit the results for uniform, normal and Zipf's distribution; but the BEMA scheduling policy also performed the best in these experiments. With CBMG distribution, the MMEMA scheduling policy shows the second best performance. However, with stable distributions (uniform, normal, and Zipf), the MMEMA scheduling policy shows lower cache-hit ratio, worse load balancing, and higher query response time than DEMA and BEMA because hotspots are less likely to move around when query arrival patterns are stable, thus MMEMA scheduling policy makes the idle servers wonder around the query space without specific direction. As a result, those servers suffer from floating EMA points that represent imprecise prediction of its cache contents, and yields low cache-hit ratio.

Next, we evaluate the scheduling policies under the dynamic query distributions. In this set of experiments, we manipulated the query distribution so that the first 10,000 queries arrive in a normal distribution and the next 10,000 queries arrive in a Zipf's distribution, the next 10,000 queries arrive in a nor-

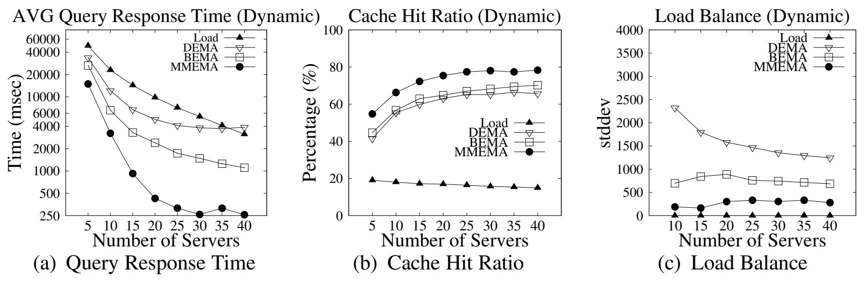


Fig. 6 *Dynamic Query Distribution (# of App. Servers)*

mal distribution with a different average, and the next 10,000 queries arrive in a uniform distribution. We adjusted the averages and variances of those distributions to create hotspots and make them move noticeably. When the query distribution is dynamic in this way, DEMA suffers from its inflexible nature and shows the worst load balance as in Figure 6(c). Because of the failure in load balancing, the DEMA policy shows high query response time and even the Load-based scheduling policy performed better than DEMA when the number of servers is 40 as shown in Figure 6(a). On the other hand, the BEMA, and MMEMA scheduling policies quickly adapt to new query patterns. It is notable that the query response time of MMEMA scheduling policy is about **180 times** smaller than the DEMA scheduling policy because MMEMA scheduling policy has much better load balance than the other EMA-based scheduling policies (Figure 6(c)) and higher cache-hit ratio than the other scheduling policies (Figure 6(b)). The cache-hit ratio of MMEMA is remarkably higher (by 10%) than the second best BEMA scheduling policy, and the average query response time is only 22% of BEMA query response time.

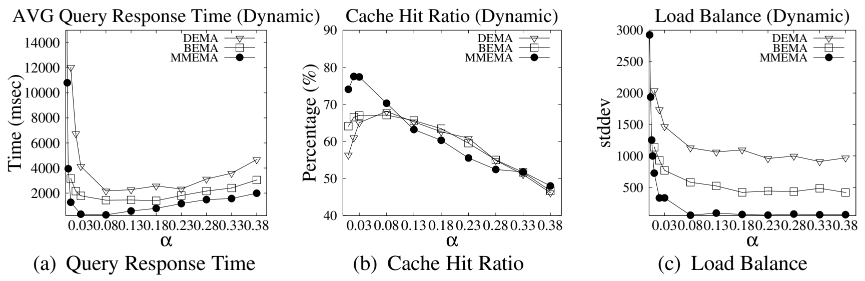
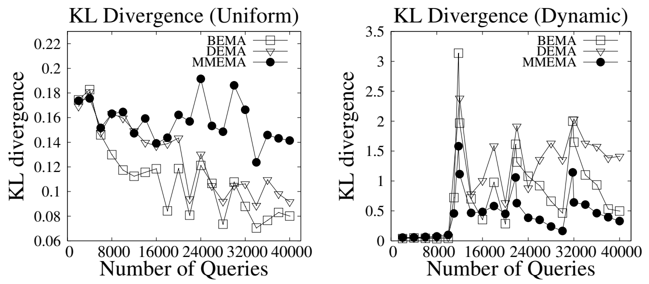


Fig. 7 *The Effect of DEMA Smoothing Factor α in Dynamic Distribution*

We ran experiments with various α values to see the effect of smoothing factor α on the system performance. In Figure 7, we observe that setting α close to 0.1 results in a low query response time for EMA-based scheduling policies. However, if the α becomes smaller, the cache-hit ratio increases but the standard deviation in the number of assigned queries also increases so that the average query response time becomes larger. Although the MMEMA scheduling policy shows good load balancing behavior, with very low smoothing factor α they also suffer from load imbalance. This is because the MMEMA scheduling policy needs to move an additional EMA point quickly to new hotspots to alleviate the heavy workload on the busy servers, but very low α values slow down the movement of the additional EMA points.



(a) Uniform Distribution (b) Dynamic Distribution

Fig. 8 *KL Divergence*

In Probability theory and Information theory, the Kullback-Leibler divergence (KL-divergence) [7] is a commonly used metric that measures the difference between two probability distribution P and Q. The KL-divergence is a logarithmic distance metric between two distributions, having zero for identical distributions and a larger value for different distributions. We employed the KL-divergence to see if the EMA points adjust their locations to form a distribution that is close to the query distribution. Intuitively, the EMA points whose locations exactly match the query distribution (i.e., more EMA points exist where more queries arrive) can perfectly balance the workload among the servers.

Figure 8 shows the Kullback-Leibler divergence values in our simulation studies that employed 100 application servers. Each application server has a cache that can hold up to 100 query results. The smoothing factor α was 0.03 for this set of experiments. Since the number of EMA points is relatively small, we partitioned the problem space into 25 equal-sized square regions and counted the number of EMA points and queries that are located inside each region. In order to measure how promptly EMA points follow current query distribution, we generated query distribution histograms - P using the most recent 2000 queries stored in a queue, and measured KL-divergence per every 200 queries using EMA distribution histograms - Q.

As shown in Figure 8, EMA-based scheduling policies make the KL-divergence smaller as more number of queries arrive. In uniform query distribution, the BEMA makes KL-divergence converge faster (as low as 0.021) than other EMA-based scheduling policies. In dynamic distribution, where hotspots change every 10,000 queries (see Figure 8(b)), the KL-divergence value goes up as high as 3.2 in the case of the three movements of hotspots. But the MMEMA policy makes the KL-divergence converge very fast by moving EMA points to the region where new hotspots are located. Note that in the second and third change of hotspots, the DEMA policy suffers from its inflexible nature, and the KL divergence value doesn't decrease below one.

5 Related Work

In the context of web server clusters, content-aware request distribution has been investigated by Aron et al. [2, 10]. They proposed the LARD (Locality-Aware Request Distribution) strategy, which assigns a request to an idle back-end server and subsequent same requests are forwarded to the same server until the server is heavily loaded. If the server is heavily loaded, LARD selects another idle server to hand off. LARD is one of the earliest distributed scheduling policy that considers both cache-hit and load balance. EMA-based scheduling policies share the goal with LARD, but our policies proactively distribute workloads to achieve better load balancing.

In the domains ranging from relational databases decision-support systems, to database middlewares designed to support data intensive analytical applications, substantial research has been conducted to optimize query execution plans and minimize the cost of processing a series of database queries [5, 11–14, 16].

Andrade et. al [1] investigated exploiting inter-query locality in distributed query processing middlewares. Their approach splits a query into parallel sub-queries, which increases the flexibility of re-arranging the query execution order and maximizes cache-hit ratio for distributed scientific data analysis applications.

In this work, we have focused on a single cluster environment, but the EMA-based scheduling policies can be employed in wide-area distributed environments as well. Zhang et al. [15] conducted a study that effectively leverages multiple back-end servers in a Grid environment. They compared the benefits of reusing a cached result with the extra overhead imposed on the server where the cached result is stored.

6 Conclusion

Leveraging distributed cache data as a valuable asset would become more and more important because the present trend is to build large scalable distributed systems by connecting small heterogeneous machines rather than purchasing expensive main frame workstations.

In this paper, we presented distributed query scheduling policies that take into consideration the dynamic contents of distributed caching infrastructure. In order to achieve load balancing as well as to exploit cached results, it is required to employ more intelligent query scheduling policies than the traditional round-robin and load-monitoring scheduling policies. Our novel EMA-based scheduling policies predict the contents of remote semantic caches as well as distributed system load information, and outperform load-based scheduling policy by orders of magnitude.

References

1. Andrade, H., Kurc, T., Sussman, A., Saltz, J.: Multiple query optimization for data analysis applicationson 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)
2. Aron, M., Sanders, D., Druschel, P., Zwaenepoel, W.: Scalable content-aware request distribution in cluster-basednetwork servers. In: Proceedings of Usenix Annual Technical Conference (2000)
3. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. Journal of the ACM 45(6), 891–923 (Nov 1998)
4. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry, Algorithms and Applications. Springer (1998)
5. Chen, F.C.F., Dunham, M.H.: Common subexpression processing in multiple-query processing. Transactions on Knowledge and Data Engineering 10(5), 493–499 (199)
6. lun Chou, Y.: Statistical Analysis. Holt International (1975)
7. Kullback, S., Leibler, R.A.: On information and sufficiency. Annals of Mathematical Statistics 22(1), 79–86 (1951)
8. Menasce, D.A., Almeida, V.A.F.: Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning. Prentice Hall PTR (2000)
9. Nam, B., Shin, M., Andrade, H., Sussman, A.: Multiple query scheduling for distributed semantic caches. Journal of Parallel and Distributed Computing 70(5), 598–611 (2010)
10. Pai, V., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W., Nahum, E.: Locality-aware request distribution in cluster-based network servers. In: Proceedings of ACM ASPLOS (1998)
11. Ren, Q., Dunham, M.H., Kumar, V.: Semantic caching and query processing. IEEE Transactions on Knowledge and Data Engineering 15(1), 192–210 (2003)
12. Roy, P., Sehadri, S., Sudarshan, S., Bhobe, S.: Efficient and extensible algorithms for multi query optimization. In: Proceedings of 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD). pp. 249–260 (2000)
13. Sellis, T.K., Ghosh, S.: On the multiple-query optimization problem. IEEE Transactions on Knowledge and Data Engineering 2(2), 262–266 (1990)
14. Xiong, X., Mokbel, M.F., Aref, W.G., Hambrusch, S.E., Prabhakar, S.: Scalable spatio-temporal continuous query processing for location-aware services. In: Proceedings of 16th International Conference on Scientific and Statistical Database Management (SSDBM) (2004)
15. Zhang, K., Andrade, H., Raschid, L., Sussman, A.: 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)
16. Zhao, Y., Desshpande, P.M., Naughton, J.F., Shukla, A.: Simultaneous optimization and evaluation of multiple dimensional queries. In: Proceedings of 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD). pp. 271–282 (1998)