GRUPD : A Scalable Update Mechanism for Distributed Hash-table Based Indexes in Computation Grid

Siddharth Rai[1]
Department of Computer Science and Engineering
Indian Inststiture of Technology Kanpur
Kanpur, India
sidrai@cse.iitk.ac.in

***Abstract*—Aggregated computing platforms, such as, computation-grid require dynamic discovery of resources satisfying a given query criteria accurately and efficiently. To simplify the task, these systems generally employ a Distributed Hash Table (DHT) index storing up-to-date attribute values of currently active resources. Nonetheless, frequent changes in attributes, like, current CPU load, available main memory, job queue length, and available network bandwidth require regular updates to DHT index and thus make efficient resource discovery a challenging task. If the resource attributes are not updated timely, already indexed values become stale and negatively influence scheduling decisions. On the other hand, updating attribute values regularly results in huge amount of network traffic and thus hamper the quality of query responses.**

To overcome these two challenges, in this paper, we present a novel mechanism to update the attribute values in a DHT index. Our proposal, named, gradual update (GRUPD), dynamically identifies computation nodes generating frequent updates. Unlike conventional update mechanism, in our proposal, these nodes send an update only if it is needed for an up-to-date scheduling decision. At the time of job scheduling, job scheduler approximates the current attribute value using an analytical model. If approximate and indexed values make different scheduling decision, the node is explicitly asked for an update otherwise node doesn't update the indexed value. Simulation based study shows that, on average our proposal is able to reduce the query response time by 11X and improve the throughput by 36X.

***Keywords*-Grid-computing; computational-grid; DHT; distributed hash-table; resource discovery, CHORD**

I. Introduction

The fundamental idea of grid computing is to aggregate computing resources and make them accessible universally through standard interfaces [15]. A grid information service plays a central role to achieve this goal. It keeps track of current resource attribute values (i.e., current CPU load, available memory capacity, current network bandwidth etc.) in a distributed data structure (i.e., distributed hash table) and exposes a simple interface to dynamically locate the resources satisfying a given query criteria. Nonetheless, massive scale of these computing platforms make the task of accurately tracking and efficiently locating these resources very challenging. If attribute values are frequently updated, it generates a huge amount of traffic and delays query responses. Where as, if attribute values are not updated timely, it effects quality of a query response and negatively influence scheduling decisions.

[1] This work was done under the supervision of Prof. Diganta Goswami while the author was a masters student at Indian Institute of Technology Guwahati, India

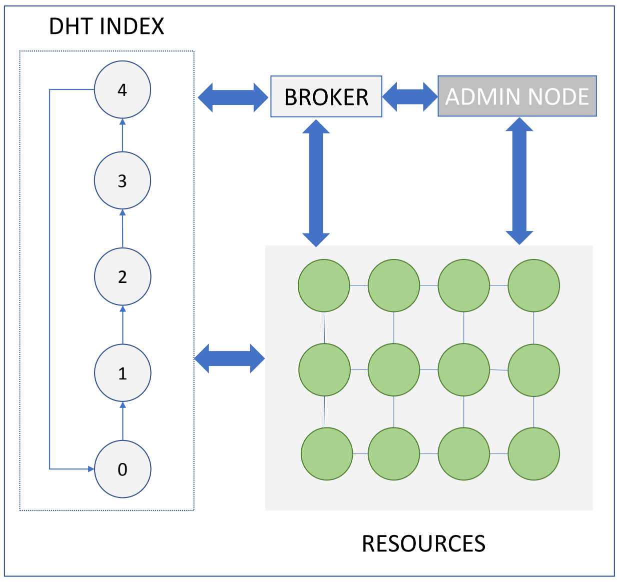

Fig. 1. Building Blocks of a Typical Computation Grid

Figure 1 shows a detailed diagram of one such system studied in this work. The system is made of four components, namely, a DHT index (also known as grid information service), computing resources, administrator (referred as admin node and helps in organizing the nodes in the system), and broker (acts as a job scheduler for grid users). The participating nodes dynamically arrange themselves in the system with the help of the admin node. Nodes part of the DHT index are responsible for tracking resource attribute values as a key:value pair. To access the index, these nodes provide a SQL like interface supporting addition and deletion operations. On the other hand, the nodes acting as a computation resources are dedicated for executing jobs submitted to the system. Whether a newly joined node will be a part of the DHT index or act as computation resource is dynamically decided by the admin node. Finally, the broker makes the system accessible to the outside world through a job submission interface. It accepts jobs from grid users and periodically schedules them to appropriate resources to maximize system utilization while meeting the job requirements.

Whenever a new node wants to join the system, it sends a join message to the admin node. Admin node assigns a role (either a DHT node or a computation resource) and sends back all the necessary information required for joining. A node serving as a DHT node can execute local job along with acting as DHT node. Whereas, a computation resource exclusively executes local or remote computation job. To remotely submit a job, grid user sends the job specification to the broker. Broker's responsibility is to find and schedule the job at an appropriate computing resource. To this end, broker periodically queries the DHT index and keeps tracks of nodes available for scheduling. As soon as enough resources become available, broker generates a schedule as per a preconfigured scheduling policy and sends the jobs to the nodes for execution. Submission and completion of jobs at a node might alter current state of resource attribute values. If the attribute value changes on job submission or completion, the new value is updated in the DHT index. In this way, DHT index is kept up-to-date of current system state. However, our experiment shows that as number of nodes in the system increases, update traffic starts interfering with the other requests (i.e., resource query by the broker) and delays their responses. Eventually, delayed query response starts effecting system utilization.

Figure 2 shows change in query time and number of scheduled jobs as system is scaled from 16 to 256 nodes. In each of these configurations, nodes joining the system remain active throughout the simulation. Out of all joined nodes, first six nodes (in temporal ordering) are given the responsibility of a DTH node and the remaining nodes act as a computation

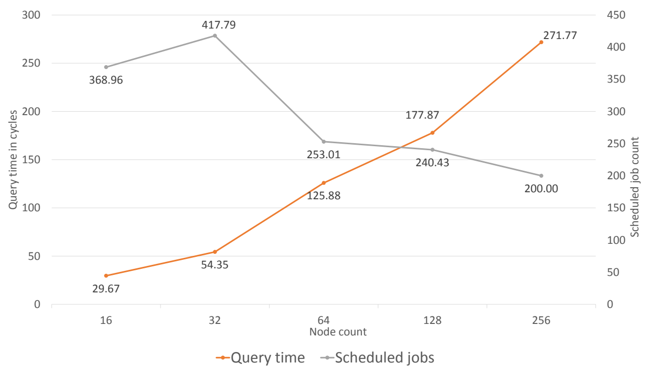

Fig. 2. Variation in query time and scheduled job count with increase in node count

resource. Broker receives remote jobs drawn from the LCG trace, whereas, computation resources execute local jobs drawn from DAS2 trace provided with The Grid Workload Archive (GWA) [23].

Broker's scheduling queue has 512 entries. In each iteration, broker schedules 48 jobs in FIFO order from this queue. To find resources to schedule these jobs, broker sends a query (i.e., $cpu - load >= 0.0$ and $cpu - load <= .50$) to the DTH index and waits for the query response to come back. The time it takes for the query response to arrive depends on current load in the system. Since, increase in number of nodes require processing more queries and updates at each DHT node, load on DHT index increases as more and more nodes are added in the system and thus effects query response time. As figure 2 shows, query time increases gradually from 30 cycles to 272 cycles as node count increases from 16 to 256. Once, broker accumulates enough computation resources, pending jobs are sent to them for execution. For a given simulation time, count of scheduled jobs depend on both query time as well as number of available resources. If query response takes longer, resource are scheduled less often. However, by scheduling more resources in each iteration, more jobs can be scheduled in the system, thus balancing the loss due to delayed responses. This trend can be seen in the figure 2, as node count changes from 16 to 32, job count improves despite the fact that query time increases from 30 cycles to 54 cycles. However, beyond 32 nodes, job count keeps degrading as node count increases. Eventually, with 256 node system, job count reduces by 84% compared to what is achieved with a 16 node system. These results indicate that system designed to update resources frequently may not always scale as node count is increased.

In this paper, we propose a novel update mechanism for the DHT index. Our proposal identifies nodes frequently generating update values using query responses seen by the broker. For such nodes, our proposal uses an analytical model to approximate their current attribute values. If scheduling decisions due to indexed and approximate values are not the same, resources are asked to generate an update.

Rest of the paper is organized as follows. Section II presents various grid information system models proposed in the literature and reviews some of the previous works that tackle scalable resource discovery problem. A motivational study demonstrating potential of controlling update traffic in improving query latency and job scheduling performance is presented in section III. In section IV, we present our proposal Gradual Update Mechanism. Finally, evaluation and conclusion is presented in section VI and VII respectively.

II. Background and Related Work

Over the years, grid research community have explored various design alternatives for grid information service architecture. These alternatives were motivated by diverse design and sharing requirements of these systems and spanned from centalized to P2P approaches. Centralized model was a popular design choice in the beginning, since, it suited small scale organizations collaborating for a scientific problem [25]. However, as these systems were scaled to much diverse set of sharing patterns, limited scalability of centralized system became apparent and thus focus was shifted to hierarchical models [5], [16], [17]. Eventually, P2P approaches were adopted to support flexible resource management, higher scalability, and improved fault tolerance [1], [2], [4], [10]–[12], [21], [22].

A. Hierarchical Models

Hierarchical models were suitable for the systems confined to small research community solving large scale problems in a collaborative environment. These systems manage local resources at Virtual Organization (VO) level [5], [16], [17]. Various such VOs connect in a hierarchical fashion to enable collaboration among VOs at different levels. Although, these systems provide better throughput and response time than centralized systems, yet, they scale only if the data they serve can be cached long enough to enjoy more hits [24], [26]. Since, dynamic resource attribute values can't be cached for a long time, these systems are limited in terms of desired scalability and response time for such dynamic environments.

B. Peer to Peer Models

P2P models address the scalability and dynamism problems faced by the centralized and hierarchical designs. These systems treat all participating nodes equally and arrange them in an overlay network (a higher level grouping of nodes). In an overlay network, participating nodes may or may not have any semantic information about the other peers. If nodes have no semantic information about the neighbours except their liveness state, the overlay is called unstructured P2P overlay. On the other hand, if nodes connect in a well defined structure (i.e., key space managed by a neighbouring node is known), they form a structured P2P overlay. Both structured and unstructured P2P systems support higher scalability and flexibility in joining and leaving the system, however, they differ in terms of guarantee given for query lookup time and employ different protocols to locate and distribute the information among participants.

1) Structured P2P Systems: Structured P2P systems use a DHT index for storing keys [4], [12], [22]. Nodes and attribute values are mapped using the same hash function onto a key space. A fixed structure, such as a ring, is used to organize nodes based on their node id. Most structured P2P systems support exact match queries in O(log N) and range queries in O(N) hops (where, N is the size of the key space). Chord [22] is an earliest example of one such structured P2P system. Nodes in Chord are organized in an one-dimensional circle according to an m-bit hash value of their node-id. Each node in Chord stores the index of all keys which fall in the range between the node and its predecessor's key. Looking up a key takes O(log N) hops and messages. The arrival or departure of a node effects at most log(N) other nodes of the index. When a new node joins the index, it takes responsibility of certain keys previously assigned to its successor. Whenever a node leaves the index, its successor takes the responsibility of all its keys. In short, with high probability, Chord ensures that each node is responsible for approximately equal number of keys and thus achieves load balancing across index nodes.

2) Unstructured P2P Systems: Unlike structured P2P system (described in subsection II-B1), an unstructured P2P systems loosely connect nodes (node can join at nay location) to form an overlay network [1], [2], [10], [11]. Gnutella and KaZaA are considered two of the most popular unstructured systems. Due to the lack of an underlying structure in those systems, prevailing resource location method is flooding. A node looking for any content, broadcasts a query into the network. All matching query responses are sent back to the originating node following the reverse path. Clearly, flooding is not scalable as it creates a large volume of unnecessary traffic in the network. To tackle this problem, each message is tagged with a Time-To-Live (TTL) field. The TTL indicates the number of hops a query can propagate. However, with this approach a query may fail to locate resource even though they are present in the system. An alternative approach to limit the amount of traffic produced due to flooding is to use dynamic query approach. In dynamic query, node initially sends a query to a subset of neighbours with a small TTL. If this attempt does not produce a sufficient number of results, node broadcasts the query again to a different set of neighbours with increased TTL. This process is repeated until a satisfactory amount of results is received, or all the neighbours are exhausted.

In summary, unstructured P2P systems are more flexible in terms of organization, but, require flooding of message to find and update current resource attribute values. Where as, structured P2P system are more rigid in their organization, but, provide guaranteed query lookup and update time. In the rest of this section, we review some of the previously proposed resource discovery algorithms for both structured and unstructured P2P systems and qualitatively compare their approaches put our contribution in perspective.

Grid information service proposals using structured P2P model take advantage of scalable query routing supported by DHT indexes. Moreover, to enable multi-attribute range queries, they make use of multiple indices (i.e., using CAN [12] and Chord [22] together) and track different attributes and their subrange in separate indices. When a multi-attribute query arrives, all indexes are looked up in parallel and responses are generated by intersecting their results. In [3] Andrzejak et al. proposed an extension of DHT based CAN system. Nodes are organized on the basis of attribute type in either a DHT or in an extended-CAN. If an attribute has limited value, a DHT is used, otherwise an extended-CAN is used. This architecture allowed supporting range queries while leveraging benefits of DHT index. In [4], Cai et al. addressed the problem of supporting multi-attribute range queries using a DHT alone. They proposed an extension of Chord [22] protocol. Multiple Chord rings are used and attributes are mapped on the ring using a locality preserving hash function. Queries are resolved in parallel on all the rings and results are intersected at the originator. Similar to [4], Spence et al. [6] proposed an extension of Pastry [14] indexing and routing scheme to resolve multi-attribute queries. To deal with dynamic attribute values, clients contact to the resources obtained to confirm their values. In a subsequent proposal, Schmidt et al. [11] presented a single DHT based approach to resolve multi-attribute query. Space-filling curves were used to map all possible dimensions to the single dimension. These transformed values are then stored in a DHT index for fast retrieval. Ratnasamy et al. [13] proposed a trie based approach to extend DHT with range search operation. Each node in trie is assigned a value range, all nodes with value in this range index value there. Each trie node is given an id corresponding to the range which are mapped to a DHT for search.

Compared to structured P2P systems, unstructured P2P systems offer a better alternative in the presence of high churn. However, due to absence of any underlying structure among nodes, these systems require more sophisticated search and update mechanism than simple lookup operation. Iamnitchi et al. [1] investigated set of request forwarding strategies for a fully decentralized unstructured P2P based grid resource discovery system. They observed that such an architecture is a good choice, if resource can be located in limited number of hops (specifically, 20 hops, considering 20ms end-to-end latency). Moreover, evaluation of different routing strategies have shown a satisfactory performance for a general distribution of resources. Another unstructured model presented in [2] proposed a two layer architecture, where the lower layer is a hierarchy of index services that publishes per-node information, and the upper layer is a P2P network to collect and distribute this information across upper layer nodes. An extension of Gnutella protocol is adopted to exchange discovery messages between nodes of the upper layer. Experiments have shown that with appropriate message buffering and merging this approach can significantly improve the search performance. To address message flooding problem in unstructured P2P systems, Puppin et al. [21] proposed a hop count based routing indices (RI). Whenever a new set of nodes join the system, their resource information is populated to their neighbours and RIs are updated. Although, RIs help improving search complexity, keeping them up-to-date generate huge amount of traffic and imposes another challenge in such design. To enable scalable routing and attribute update, Marzolla et al. [10] proposed a novel tree vector organization of nodes and bit-indices based routing mechanism. Their proposal takes advantage of low-diameter tree network and sparse routing indices to efficiently route resource discovery messages. Empirical studies show that query radius grows as $O(\log(N))$, while incurring constant number of updates. However, query still spans as $O(N)$. Moreover, for bit-indices may not be able to adapt with resource diversity.

III. Motivation

In this section, we motivate our proposal by showing that delaying updates indeed reduces the amount of contention in the DHT index. However, at the same time, delaying update values increases the job to node mismatch due to stale attribute values present in the index. This mismatch causes more jobs to wait at the node before start of their execution and thus negatively influences system utilization.

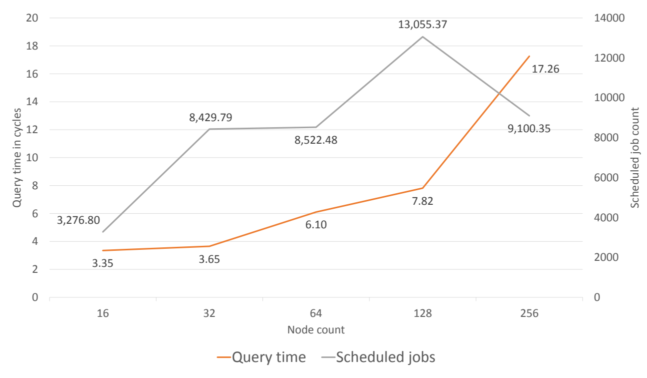

Fig. 3. Variation in query time and scheduled job count with increase in node count when updates are not sent to DHT index

Figure 3 shows the number of scheduled jobs and resultant query latency when attribute updates are delayed. As can be seen in the figure, reduction in update traffic improves the query latency by more than 15X; resulting in 45X improvement in scheduled job count.

However, when update messages are delayed, scheduling decisions are made out of stale attribute values. As a result, jobs may not be scheduled on the most adequate resources. Figure 4 shows the number of jobs that had to wait at the node before commencing their execution. As the figure shows, with reduction in update traffic, jobs waiting for resource to become available starts increasing. This delay degrades overall system utilization compared to when jobs are distributed across all available resources.

These results indicate that to be effectively, updates should be controlled such that scheduling decisions are not negatively effected.

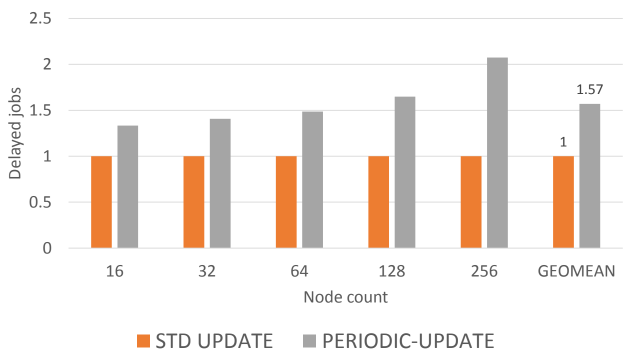

Fig. 4. Count of waiting jobs when updates are delayed

One way to achieve this goal would be to provide the scheduler access of the attribute values required to make an optimal scheduling decision. While, the updates that may not effect the scheduling decision can be delayed to reduce the update traffic. These two objectives form the basis of our proposal presented in section IV.

IV. Gradual Update Mechanism

In this section we present our proposed gradual update mechanism for updating a DHT index. Our proposal identifies a set of nodes updating their attribute values frequently. These nodes are explicitly asked to send the update only when the update can alter current scheduling decision. There are three components in our design. 1) A mechanism to identify nodes generating large number of updates 2) An analytical model to approximate resource attribute values to decide if the indexed value needs to be updated 3) Mechanism to request an update from the resource. In the rest of the section we will present each of these component in more detail.

Identifying Nodes Generating Frequent Updates: The DHT index in our system is updated by the participating nodes in a P2P fashion as a result nodes can not determine whether or not a new update will influence the scheduling decision. To overcome this shortcoming our proposal track the last seen attribute values for each node at the broker. We take advantage of the fact that the broker queries the index before making a scheduling decision. We track the current attribute values in an update tracking table shown in table I.

For each type of attribute for a node, we track four pieces of information: ID, RANGE, LAST VALUE, and GRUPD. Where the ID stores per-node resource id, RANGE is a sub-range of attribute values seen so far, VALUE is the last seen attribute value, and the GRUPD is a flag that is set if update is to be explicitly asked. Out of these, RANGE and LAST VALUE together determine the changes happening to an attribute value. RANGE is update on receiving a query response as follows. If attribute value obtained in the query is not the same as the LAST VALUE, the range is increased to include the new value. On the other hand, if the value is same as that of the LAST VALUE, the range is halved such that the new value still remains with in the range. Periodically the value if RANGE field is checked and if it contains more than one value, GRUPD bit is set to 1 and resource is notified for on-demand update.

Approximating Attribute Values: The GRUPD bit only identifies the nodes that frequently generate an update. However, to be able to reduce the count of updates from these nodes, we approximate attribute values using a queuing model. As a result we eliminate the need for and update if approximate value can be used to make scheduling decision. As soon as it is identified that the approximate and indexed value may lead to different scheduling decisions, approximation for the resource is terminated.

$$Pending\ Jobs = JOB\ RATE \times LATENCY \quad (1)$$

$$Current\ Value = \frac{PENDING\ JOBS}{BANDWIDTH} \quad (2)$$

Equation 1 and 2 show the model used to obtain the approximate values. The model depends on three input parameters. First, the job arrival rate for a resource, second, average job completion time, and finally, the node bandwidth. Broker keeps track of the rate at which jobs are scheduled at a node and the average time to complete a job. Node bandwidth is a static parameter and known in advance. These parameters are substituted in equation 1 and 2 and approximate values are obtained.

Requesting Attribute Update: Figure 5 presents the algorithm for requesting explicit attribute updates. As shown in the figure, for a resource if GRUPD

TABLE I
UPDATE TRACKING TABLE

ID	RANGE	LAST VALUE	GRUPD
1	(A1, B1)	A1	1
...	...	...	...
2	(A2, B2)	B2	0

flag is found to be set and as per approximate attribute value job should not be scheduled on that resource, update is requested and resource is purged from the update tracking table.

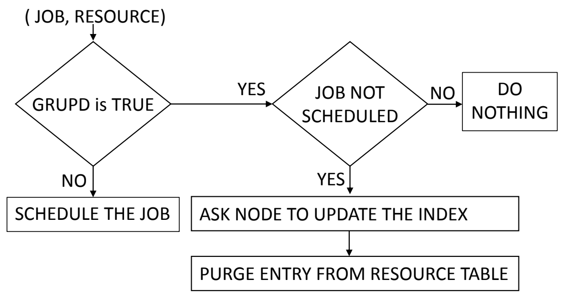

Fig. 5. Algorithm to request update of current resource value

V. SIMULATION ENVIRONMENT

We evaluated our design using an in-house java based discrete-event simulator. Our simulator models a computation grid system similar to the one shown in figure 1. We use SimJava [9] discrete event simulation library for scheduling and processing events while the grid resources are modelled using the GridSim [8] grid simulation library. Each node in our system can be dynamically configured to act either as a computation or a DHT index node. There are two special nodes in our system. The first node is the admin node which helps in organizing the rest of the nodes of the system. The second node is the broker node whose job is to schedule remote jobs on an appropriate compute resource. Each of these nodes run a stub routine to carry out the task it is configured to execute. For example, a node acting as a DHT index node executes a stub routine to index an attribute value and to perform a query over the DHT index. When a new event is received, corresponding stub routine is invoked to process the event and if required, a new response event is sent back to the requester. In this way, all participating nodes interact with each other to accomplish their tasks.

Each node in our system has four quad-core CPUs, 4 GB RAM, and a 1 Gbps network link. All nodes execute job traces drawn from the Grid Workload Archive [23] for both local and remote jobs. Table V provide a detailed system configuration and the list of workloads used for the evaluation. Local jobs at a node are executed as and when they arrive. Where as the remote jobs are submitted by the grid users and are scheduled on the resources by the broker. Broker

TABLE II
SYSTEM CONFIGURATION

System configuration		
Resource	*Configuration*	
	No of sites	16-256
	CPU per site	4
	CPU cores	4
	Resource type	uniform
	Attributes	
	CPU speed	4GHz
	Network bandwidth	1 Gbps
	Dynamic attribute	load
Workload	Remote jobs	LCG [23]
	Local jobs	DAS-2 [23]
Simulation time	Until the completion of trace	
Scheduling algorithm	Best-Fit with 512 entry job queue	

tracks all ready to schedule jobs in a 512 entry job queue. As soon as pending job count exceeds 48 or a scheduling period expires, broker queries the DHT index to find the currently available resources and schedules the jobs on to the best matching resource.

VI. EVALUATION

In this section we present the evaluation of our proposed solution in terms of achieved throughput, resultant query latency and the count of jobs waiting before commencing their execution. We have compared our results with two existing proposals. The first proposal is our baseline (labelled as STD UPDATE in figures), where the updates are generated with change in attribute values. Where as, the second solution (labelled as PERIODIC UPDATE in figures) sends an update periodically and thus reduces the number of updates in the system. Our proposal is labelled as GRUPD in figures.

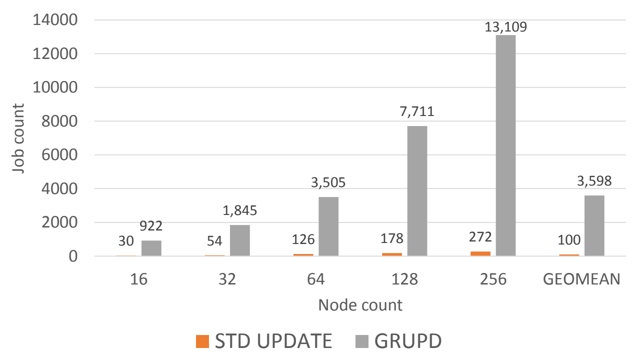

Fig. 6. Improvement in job count as nodes are scaled from 16 to 256

Figure 6 shows the improvement in scheduled job count for both the baseline and GRUPD. Since, in the baseline, an attribute value is updated as and when it changes; it results in huge amount of update traffic. The x-axis and y-axis in figures show the node count and the number of jobs scheduled corresponding to each node count for each of the proposals. As the figure shows, GRUPD is able to improve the scheduled job count by 36X, on average. Moreover, as the number of nodes in the system is increased, scheduled jobs are able to scale proportionally achieving 48X improvement for a 256 node system.

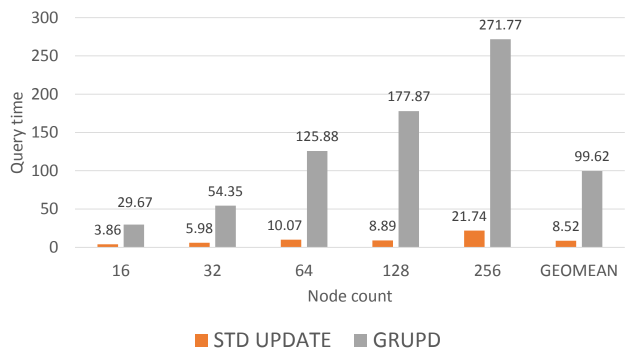

Fig. 7. Improvement in query time compared to standard update mechanism

One of the primary reason for our proposal to achieve a better scalability with increase in node count is the reduction in query response time. Figure 7 shows the query time as node count is scaled from 16 to 256 for the baseline and our proposal. On average, we are able to improve query time by 11X with the maximum improvement of 20X for a 128 node system. It is interesting to note here that a 256 node system achieves about 12X improvement, which is approximately half of what is achieved by a 128 node system. Nevertheless, a 256 node system

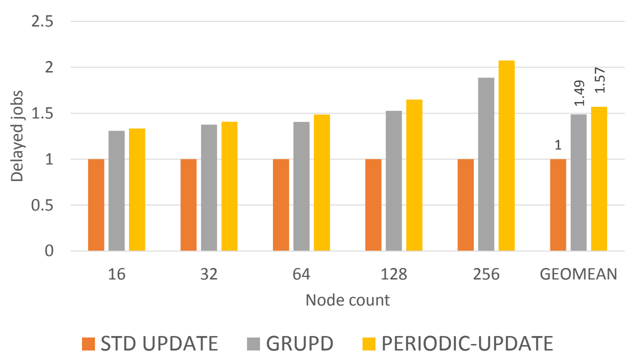

Fig. 8. Change is waiting jobs at the scheduled node as nodes are scaled from 16 to 256

is able to achieve much better scalability. This result reinforces the observation made in section I that reducing query response time is necessary along with increase in node count for a scalable DHT index.

However, as we have shown in section III, another important factor effecting overall system utilization is the reduction in job to node mismatch. Figure 8 shows the increase in waiting jobs as number of nodes are scaled up to 256. As the figure shows, on average, we are able to reduce such jobs by 8% compared to a system where attributes are updated as per PERIODIC-UPDATE policy. However, this improvement is much lower than what we expected and we are investigating this further in our future work.

VII. Conclusion

In this paper, we have presented a novel update mechanism for DHT based indexes. Our proposal dynamically identifies the nodes generating bulk of update values. These nodes are informed to update the attributes only if the update can influence the scheduling decision. As a result, we are able to significantly reduce the amount of update traffic. On systems with 16 to 256 compute nodes, on average, we are able to improve the system utilization and query response time by 36X and 11X, respectively. Moreover, compared to a setup with periodic attribute updates, we are able to reduce the average waiting job count by 8%.

References

[1] A.Iamnitchi, I.T.Foster, A Peer-to-Peer approach to resource location in Grid environments, J.Weglarz, J.Nabrzyski, J.Schopf, M.Stroinski (Eds.) Grid Resource Management,Kluwer,2003.

[2] D.Talia, P.Trunfio, Towards a synergy between P2P and Grids,IEEE Internet Computing 7(4),2003 ,pp. 94-96

[3] A.Andrzejak, Z.Xu., Scalable efficient range queries for Grid information services, in Proc.2nd international Conference on Peer-to-Peer Computing,P2P 2002, pp.33-40.

[4] M.Cai, M.Frank, J.Chen, P.Szekely, MANN:A multi-attribute addressable network for Grid information services, in:Proc. 4th Int. Workshop on Grid Computing,GRID 2003, pp.184-191.

[5] C.Mastroianni, D.Talia, O.Verta, A super-peer model for resource discovery services in large scale Grids, Future Generation Computer systems21(2005), pp.1235-1248.

[6] D.Spence, T.Harris, XenoSearch:Distribued resource discovery in XenoServer open platform. in: Proc.Twelfth IEEE Int. Symposium on High Performance Distributed Computing.HPDC-12, 2003, pp.216-225.

[7] A.R. Bharambe, M.Agrawal, S.Seshan, Mercury:Supporting scalable multiattribute range request,in: Proc ACM SIGCOMM 2004 conf. on Applications.Technologies, Architectures and Protocols for Computer communications, 2004 pp.353-366.

[8] Buyya, R., Murshed, M.M. (2002). GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency and Computation: Practice and Experience, 14, 1175-1220.

[9] Fread Howell, R. McNab, Simjava: A Discrete Event Simulation Library For Java. (1998).

[10] M.Marzolla, M.Mordacchini, S.Orlando, Peer-to-Peer Systems for discovering resources in a dynamic grid, Parallel Computing 33(2007) 339-358.

[11] C.Schimidt, M.PArashar, Flexible information discovery in decentralized distributed systems,in:Proc 12th Int. Symp.on High-Performance Distributed computin,HPDC-12 2003,pp. 226-235.

[12] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. 2001. A scalable content-addressable network. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01). ACM, New York, NY, USA, 161-172.

[13] S.Ratnasamy, J.M. Hellerstein, S.Shenker, Range queries over DHTs IRB-TR-03-009,Inel Corporation,2003.

[14] Rowstron A., Druschel P. (2001) Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Guerraoui R. (eds) Middleware 2001. Middleware 2001. Lecture Notes in Computer Science, vol 2218. Springer, Berlin, Heidelberg.

[15] Ian Foster, Carl Kesselman, Steve Tuecke, The Anatomy of the Grid, Intel journal of supercomputing applications,2001

[16] Hélène N. Lim Choi Keung, Justin R. D. Dyson, Stephen A. Jarvis, and Graham R. Nudd. 2003. Predicting the Performance of Globus Monitoring and Discovery Service (MDS-2) Queries. In Proceedings of the 4th International Workshop on Grid Computing (GRID '03). IEEE Computer Society, Washington, DC, USA, 176-.

[17] K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman, "Grid information services for distributed resource sharing," Proceedings 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, CA, USA, 2001, pp. 181-194. doi: 10.1109/HPDC.2001.945188

[18] Warren smith, Abdul waheed, Davis Meyers, Jerry yen An evaluation of alternative design of grid information service, IEEE Internet computing 2003

[19] Rajkumar buyaa, Mazoor mursheed, GridSim:toolkit for modeling and simulation of distributed resource management and scheduling for grid computing, concurrency comutat:pract. Exper. 2002, pp.1175-1220

[20] P.Trunfio, D.Talia,H.Papadakis, P.Fragopolulou, M.Pennanen, K.Popov, V.Vlassov, S.Haridi, Peer-to-Peer resource discovery in Grids:Models and systems, Future generation computer systems 23(2007), pp.864-878

[21] D.Puppin, S.Moncelli, R.Baraglia, N.Tonelotto, E.silvestri, A Grid information service based on Peer-to-Peer, in Proc. 11th Euro-Par conf.EuroPar 2005 in:LNCS vol.3684 Springer, 2005, pp.454-464.

[22] I.Stoica, R.Morris, D.R. Arger, M.Frans Kaashock, H. Balakrishnan, Chord:A scalable Peer-to-Peer lookup service for internet applications,in Proc:ACM SIGCOMM 2001 Conf on Applications, Technologies,Architecture,and Protocols for Computer Communication, 2001, pp.149-160

[23] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, Dick H.J. Epema, The Grid Workloads Archive, Future Generation Computer Systems, Volume 24, Issue 7, 2008, Pages 672-686, ISSN 0167-739X, https://doi.org/10.1016/j.future.2008.02.003.

[24] Zoltan Balaton, Gábor Gombas, Zsolt Németh, 2002, A Novel Architecture for Grid Information Systems, In Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '02). IEEE Computer Society, Washington, DC, USA

[25] Litzkow M., M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations". In Proceedings of the 8th International Conference of Distributed Computing Systems, pages 104–111, June 1988.

[26] Xuechai Zhang, J. L. Freschl and J. M. Schopf, "A performance study of monitoring and information services for distributed systems," High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on, Seattle, WA, USA, 2003, pp. 270-281. doi: 10.1109/HPDC.2003.1210036