Analysis of Query Matching Criteria and Resource Monitoring Models for Grid Application Scheduling

Ronak Desai, Sameer Tilak, Bhavin Gandhi, Michael J. Lewis and Nael B. Abu-Ghazaleh
State University of New York, Binghamton NY 13902
{rdesai, sameer, bgandhi, mlewis, nael}@cs.binghamton.edu

Abstract

Making effective use of computational Grids requires scheduling Grid applications onto resources that best match them. Resource factors (e.g., load, availability, and location), and demand factors (number and distribution of application resource requests) influence scheduling decision success. The scale of the Grid makes maintaining detailed up-to-date information regarding all resources impractical. Thus, concurrent distributed schedulers must attempt to make scheduling decisions based on dynamic and potentially inconsistent resource state information. In this paper, we evaluate the effect that the criteria for selecting scheduling matches has on the success of scheduling decisions. We focus on three criteria: information freshness, resource distance from requesters, and past behavior; we evaluate the quality of the schedule for various resource monitoring models, Grid loads, and Grid overlay topologies. Among our findings is the counter-intuitive result that favoring freshness can sometimes hinder overall system performance; a combination of resource distance and past scheduling success performs best. We also evaluate a pure resource state pull model with caching, and demonstrate that proactively pushing dynamic state information to schedulers is beneficial with respect to several different evaluation metrics.

I. Introduction

Realizing the potential of Grids requires effective middleware services that manage the complexity of the environment and present useful abstractions to applications. Among these services is Grid application scheduling, which matches application resource requests to resources that can best meet them. This problem is challenging because the scheduling has to be carried out in a large scale, distributed, heterogeneous, and dynamic environment. Many concurrent and possibly competing resource reservation requests are submitted to different schedulers, causing resource states to vary significantly in short periods of time.

To make appropriate decisions, schedulers need up-to-date and accurate information about Grid resources. This information can be made available using either the *push model* to periodically distribute information out into the Grid, or the *pull model* to collect information directly from resources on demand, to satisfy specific requests.

The push model may not directly lead to application requests being satisfied, and may cause schedulers to use stale (i.e. old and potentially inaccurate) information. On the other hand, the pull model requires significant delay at application run-time while the remote resource information is collected. Moreover, in terms of overhead, the pull model incurs the cost of resource discovery with each query. In contrast, the push model requires a constant overhead as each resource provider periodically advertises its resource availability, and the cost of this advertisement may be amortized over multiple queries that use it. Caching may improve the pull model performance, but may also lead to the use of outdated information, and scheduling may fail without pulling the remote resource information again. In Section II, we describe more specific background and related work for Grid resource discovery and query matching.

An underlying operation needed by resource monitoring algorithms is a protocol for *dissemination* of resource data to multiple schedulers distributed across the overlay network. Structured approaches, such as multicast, require significant overhead to maintain the multicast backbone, and make it difficult to disseminate information at different granularities. Moreover, brute force approaches such as flooding lead to excessive overhead. In previous work, we studied the use of efficient probabilistic forwarding algorithms for disseminating resource information non-uniformly. More specifically, resources send their information to update schedulers with a frequency and granularity that is inversely proportional to the distance between the resource and the scheduler (alternatively, the same algorithms can be used to send queries non-uniformly from schedulers to resources in the pull model). This model is described in more detail in Section III. The first contribution of this paper is to analyze the effectiveness and overhead of different resource monitoring models (push, pull, and variants of each) using different network-wide dissemination algorithms.

Resource monitoring algorithms make available to schedulers information regarding multiple matching resources. The second contribution of this paper is to explore the following question: how should schedulers select which resource to schedule a query to, and what effect does this query matching policy have on the performance of the scheduler? We explore several criteria for ranking the resources, including the distance of the resource, the freshness of the information, the historical scheduling success to the resource, as well as combinations of these factors. We evaluate the effect of these factors on the scheduling success of different resource monitoring algorithms (including the effect of the dissemination algorithm), overlay topologies, dissemination rates, offered application loads, and types of available resources.

Section IV experimentally evaluates the proposed query matching policies under different resource monitoring models. Our main high-level observations are that the push approach reduces scheduling overhead and improves overall query success. However, within satisfied queries, pull results in slightly better schedules, because it acts on fresh information. With query matching, push performance approaches pull performance, even in this respect. Furthermore, using freshness can unexpectedly lead to worse schedules because it causes contention on the resources that most recently advertised their presence; the best matching criteria combines hop count and historical scheduling success. We summarize contributions and discuss future work in Section V.

II. Related Work

Our previous work introduced and characterized information dissemination protocols; we designed and simulated non-uniform protocols that result in increased information quality available at nearby nodes [1]. This helps keep dissemination overhead low, while maintaining acceptable accuracy of information at places where needed. In followup work, we showed that probabilistic dissemination protocols have complex coverage characteristics and are not easily controllable. We proposed hybrid protocols to enable better control over dissemination coverage [2]. This paper integrates the dissemination protocols into a resource monitoring and scheduling simulation framework. We significantly extend our previous work by studying and evaluating resource monitoring models (push vs. pull). We also study for the first time the influence of query matching criteria on scheduling success.

Iamnitchi et.al. [3] proposed strategies for query processing and evaluated their scheduling success using average hop count between querier and resource. Our study includes this metric, in addition to the percentage of queries satisfied, and the potential application startup overhead of preprocessing and resource reservation. We encapsulate their "best neighbor" heuristic in our *confidence factor* ranking function component and consider the effect of two additional heuristics in ranking—distance between the requester and the resource, and the freshness of the disseminated state information. Finally, whereas their approach is purely pull-based, we study the effect of push-based dissemination as well, and explore the interaction with the underlying dissemination algorithms.

Another decentralized resource discovery approach [4] uses reservations with two different matching schemes—best turn-around time and closest attribute match. In our work, a requester matches queries locally to find a candidate set of nodes that match all the query requirements, and then ranks this set. Our work also differs in that it does not require exclusive resource reservation.

In the Flock-of-Condors approach [5], *Condor pools*–organized in a P2P structure using the Pastry routing protocol–disseminate resource information and sharing policies (collectively called ClassAds [6]) to neighbors. Pools contact one another to negotiate this resource sharing. Thus, this approach uses a combination of push and pull. The authors use turn-around time as the primary performance metric. In contrast, we study various resource monitoring models and analyze the effects of query matching policies on several evaluation metrics, in a simulated environment.

Grid Information Services [7] require resources to be registered with the MDS directory service. Clients query this service to obtain information about current resource status and sharing policy. Directory server organization, policies for information dissemination, and resource selection criteria are left unspecified.

Lv et.al. [8] propose random walks as an alternative to query flooding in unstructured P2P networks; *k-walker* random walks help locate objects at lower overhead than TTL-scoped flooding [9]. This approach relies on active object replication in the overlay. Our search must consider dynamic information, not just static objects, but could potentially incorporate random walks to improve dissemination.

III. Resource Discovery Model

We model a computational Grid as a set of nodes[1] organized in an overlay topology. Each node acts as a resource provider that accepts and runs applications from requesting clients within the Grid, and also as a scheduler that may need to find resources for queries that could not be locally satisfied. Each node is characterized by a node descriptor tuple (T, U, S), where T refers to the Type of resource (e.g. a cluster or a supercomputer), U refers to the available resource Units (e.g. a 32-node cluster might contain 32 "units") and S refers to the resource's available time Slots (e.g. a four hour block of time on a cluster node). This characterization can be extended to represent heterogeneous resources at a node as a vector of the resource types and information. A query (or resource request) is characterized by the same three parameters contained in a node descriptor. A query contains the requested resource type (T), the required number of resource units (U) and required time slots (S). The use of U allows us to model situations in which more than one request uses a resource simultaneously (this is different from other reservation schemes [4]). We

[1]Note that a node in our description corresponds to an aggregation of one or more homogeneous resources such as a Condor pool with a Centralized Manager or a cluster with a designated Cluster Manager.

use S to model application-specific time requirements, for example, resource reservation for certain time periods. Note that U and S are described relative to a normalized standard and can be re-mapped locally to the existing hardware capabilities.

A. Architecture

The two main architectural components of our approach are: (i) resource monitoring: this is the process of collecting resource information for use in scheduling decisions. It can be implemented via models such as push and pull. In either model, a dissemination function is needed to "broadcast" resource information (push model) or resource queries (pull model) to other nodes in the network; (ii) query matching and resource ranking: this component determines in what order to request the matching resources for a given query.

We explore both the pull and push models of resource monitoring in our study. More specifically, in the push model, the information repositories are replenished proactively by state information disseminated from the resource. Alternatively, in the pull model, information is requested explicitly by disseminating a request from a scheduler to the resources (caching may be used to reuse collected information or information requested by other nodes). One of the contributions of the paper is to evaluate these resource monitoring approaches under different conditions and against different dissemination schemes.

We evaluate the Biased and Unbiased protocols developed in earlier work [1] as dissemination primitives. These protocols use a probabilistic approach to disseminate information more frequently to the nearby nodes than the remote ones; thus, they capitalize on the intuition that most queries are best matched to resources that are nearest to them. In the Biased protocol, the dissemination (or forwarding) probability at an intermediate node is inversely proportional to that node's distance from the source node. The Unbiased protocol is simply gossipping: intermediate nodes forward the information with constant probability P. We refer interested readers to [1] and [2] for more details; these works characterize the dissemination process but do not address the query matching component of the system, which we explore in this paper. As a result of the dissemination, a subset of the total nodes collect the disseminated information in their local repositories (push model) or receive the request (pull model).

In general, the resource monitoring algorithm leads to knowledge of multiple resources that can support a received query. In the query matching component of the model, when a query cannot be satisfied locally, a node consults its local repository. Specifically, a generated query's attributes are matched against the attributes of the nodes in the local repository. We use "equality" match for the resource type attribute, and greater-than-or-equal-to match for resource units and time slots; for example a node providing 20 slots can match a requirement for 10 slots. All such candidate nodes serve as inputs to a ranking function, which orders them in decreasing order of their "fitness" for the scheduler.

B. Ranking Heuristics

We consider the following ranking heuristics:

Hop Count (HC) captures the distance between a query generator and a resource node along the overlay topology (as received from the resource state push). Scheduling a job near its "launch point" reduces data transfer time and startup costs. Therefore, each node in the local repository is assigned a Hop Count value in inverse proportion to that node's distance from the query generator.

Freshness (FR) indicates how up-to-date the information is, at the time of resource selection, using timestamps. We use this heuristic expecting that fresh information about resource availability is more likely to be accurate and reliable.

The ***Confidence Factor (CF)*** heuristic captures a node's experience with the nodes it scheduled to in the past. We expect that queries are more likely to be satisfied by a node that previously satisfied similar types of queries. A scheduler's confidence in a node increases when the node is successfully reserved, and decreases when it denies a reservation.

We use these three heuristics as components to derive an overall rank for each resource provider, using the following weighted average formula:

$$Rank_{AB} = W_1 \frac{1}{HC_{AB}} + W_2 \frac{1}{FR_{AB}} + W_3 CF_{AB} \quad (1)$$

$Rank_{AB}$ is an overall rank calculated at the query generator A for resource provider B. HC_{AB} is the Hop Count distance between A and B, FR_{AB} represents the freshness of information of B at A at ranking time, and CF_{AB} indicates node A's confidence in B. This ranking formula can be instantiated with different combinations of W_i, we present results for a handful of interesting possibilities.

C. Resource Discovery Algorithm

The query matching and resource reservation components of the scheduler proceed as follows. First, the scheduler consults a local repository to identify the full set of candidate nodes that match the given query's resource requirements. The scheduler applies the ranking formula described above to each node in the candidate set, to create an ordered list of candidates. A Reservation Request is then sent to the first node in the list; the request specifies the query requirements, including the resource type, number of resource units, and required time slots. The receiving node matches the query requirements against its current state as described by its local (T, U, S) tuple, to determine whether it can accommodate this request. If it can, the node returns a Reservation Accepted message. If the request cannot be satisfied, the node sends a Reservation Denied message, which removes the selected node from the scheduler's list. The scheduler then proceeds similarly through remaining candidate nodes. In the event of getting Reservation Denied message from all the candidate nodes, resource reservation component notifies the scheduler so that it can resubmit the

request at a later time. We however, do not enforce any specific policy regarding request re-submission.

IV. Experimental Evaluation

We use the Scalable Simulation Framework Network (SSFNet) for all experiments [10]. The GT-ITM topology generation tool [11] generates all simulation topologies, which we then convert into a Domain Modeling Language (DML) schema for use with SSFNet.

In our setup, a simulation cycle is 125 simulation seconds; for the push model, at the beginning of each simulation cycle, nodes disseminate resource information using non-uniform protocols. We evaluate the performance of various ranking functions across different non-uniform protocols, including Unbiased (with probabilities 0.2, 0.5 and 0.8) and Biased protocols. Five seconds after the dissemination, all the nodes generate queries at random time offsets within a 10 second interval. For consistency purposes, in the pure pull case, the nodes generate queries at a random time between 5 to 15 seconds from the start of the simulation cycle. Generated queries are then either forwarded (in the pull case) or matched with resource information in the local repository (for push or pull-with-caching). The information, queries, responses, reservation requests, and reservation replies are forwarded through the overlay topology using shortest path routes, which are maintained and updated at each node through the push process. For the push model, the first few cycles in the simulation are "warm-up cycles" consisting only of information dissemination to populate local repositories. We do not use such warm-up cycles for pure pull simulations.

The confidence factor for each node starts at 0.5. When a resource is successfully reserved, that provider's confidence value is increased by 0.05; when a request is rejected, it is decreased by 0.05. Confidence values are bounded below by 0 and above by 1. We measure information freshness in terms of simulation cycles rather than simulation clock seconds. Therefore, all information received within the same simulation cycle is considered equally fresh. The three ranking values are normalized between 0 and 1 to allow their weighted combination. When we consider multiple factors, we give each of them equal weight. In the following experiments, all nodes provide resources with the ten initial resource units and 1000 resource time slots. Each node is assigned one of three different resource types using a uniform random distribution. All nodes generate one query in each cycle, with resource unit and time slot values selected randomly between 1 and 5. This study considers only dedicated Grid resources that are available throughout the simulation, not resources that join or leave the Grid dynamically. Upon being reserved, a resource's available units are reduced by the amount specified in the query, and increased when the reservation ends.

In section IV-A, we study the ranking functions' effect on performance metrics by varying the information dissemination rate. We compare pull and push approaches, and demonstrate the benefits of ranking with push. We study and analyze the HC, CF-HC and CF-HC-FR ranking combinations across different dissemination protocols. Our results indicate, for example, that using confidence with hop count (CF-HC) performs reasonably well with respect to all performance criteria. We also make an interesting observation that despite the intuitive usefulness of freshness in ranking, inclusion of freshness in ranking adversely affects the overall performance when the information dissemination rate is less than the query generation rate. In Section IV-C, we study ranking under different system load scenarios.

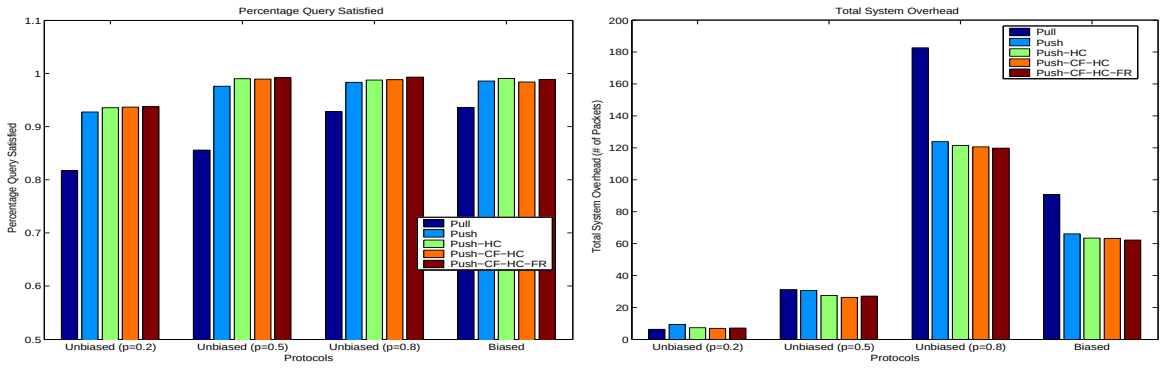
In Section IV-D, we optimize the pull approach by caching responses, and present simulation results comparing pull-caching with the push-ranking approaches. We observe the expected inherent trade-off between overhead and the percentage of queries satisfied. We show that the push-ranking approach is more controllable compared to the pull approach. Push-ranking takes a *resource-centric* view and allows the resource provider to control the tradeoff between system overhead and the percentage of queries satisfied. More specifically, a resource provider can reduce its dissemination frequency to match the required query satisfaction. Therefore, we believe that push-ranking distributes control between providers and requesters, which is not possible using a pure pull approach.

A. Study of Ranking parameters

Here we vary the information dissemination rate and study its effect on the performance of several ranking functions. We present simulation results for a transit stub type random topology of 600 nodes.

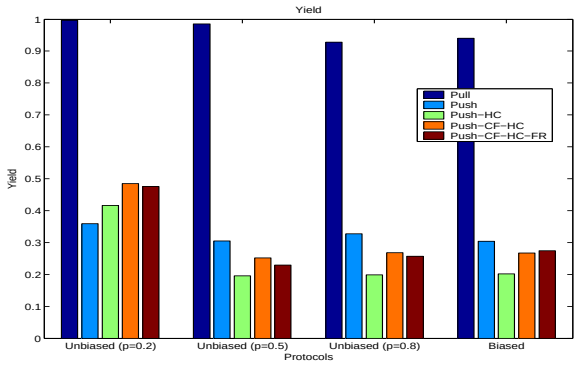
Figure 1 shows results for the case when the information dissemination rate is equal to the query generation rate. That is, each node disseminates information and generates one query in each simulation cycle. Using information dissemination increases the percentage of satisfied queries compared to the pure pull approach. Specifically, for the Unbiased 0.2 protocol, the percentage of queries satisfied is increased by 11%, as shown in Figure 1(a). Furthermore, the total system overhead is reduced for Unbiased 0.5, Unbiased 0.8 and Biased protocols. Figure 1(b) shows that up to 32% reduction in overhead is achieved in the case of Unbiased 0.8. In this case, even though ranking improves query satisfaction by merely 1 or 2% (marginal improvement), it reduces total system overhead. For example, Figure 1(b) shows that Unbiased 0.5 with a CF-HC ranking function results in additional reduction of up to 14%[2] compared to pure push. Figure 1(d) shows that using ranking functions helps find resources in the vicinity of query generators. Figure 1(c) shows that yield (the inverse of the number of reservation requests per satisfied query per query generating node) is decreased compared to push when ranking functions are used, because ranking functions cause aggressive search for resources near the query generator. Figure 1(c) shows that the performance of two ranking functions, CF-HC and CF-HC-FR, are comparable, but HC alone doesn't perform well in terms of yield, compared to CF-HC.

[2]We derived percentage numbers from actual numerical data obtained from simulations.

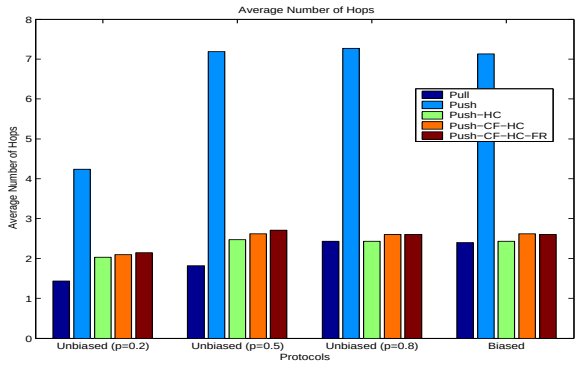


(a) Percentage Query Satisfied

(b) Total System Overhead



(c) Yield



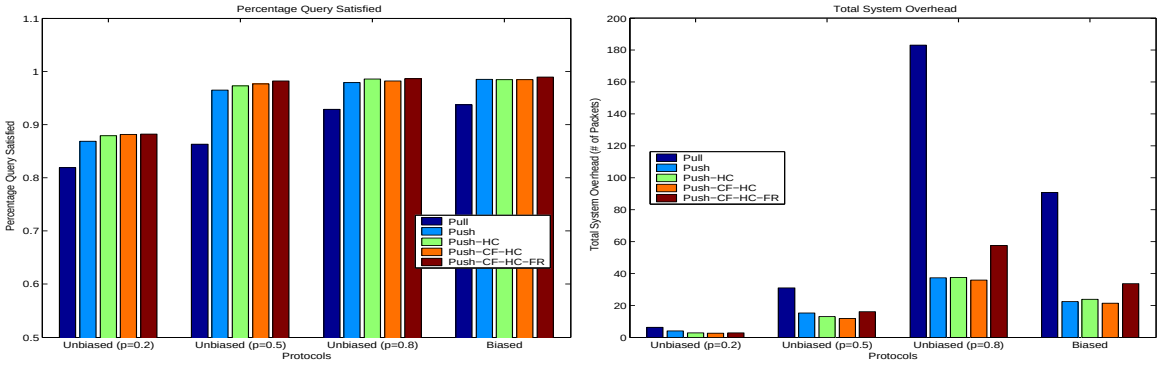
(d) Distance

Fig. 1. Dissemination rate = Query generation rate

Figure 2 shows results for the case when the information dissemination rate is $(1/5)^{th}$ of the query generation rate. In this case, nodes are divided into 5 disjoint groups; the first group disseminates information in simulation cycles 1, 6, 11 and so on, while the second group disseminates information in simulation cycles 2, 7, 12, etc. However, each node generates a query in each simulation cycle.

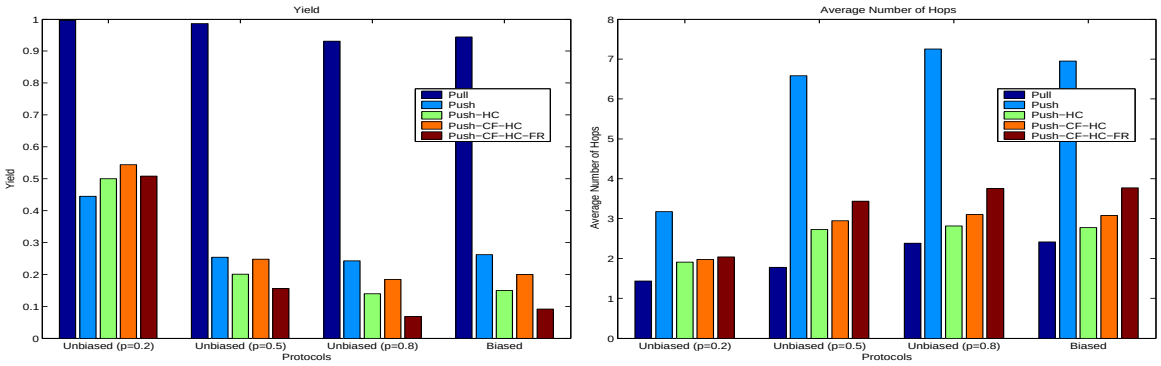
Figure 2(a) shows that push protocols result in higher percentages of queries satisfied, compared to pull. Unbiased 0.5 increases the percentage of queries satisfied by 10%. Furthermore, the system overhead for the push approach is 35% to 80% less than pull. Using any ranking function performs as well as pure push in terms of the percentage of queries satisfied (Figure 2(a)). Figure 2(b) shows that the CF-HC ranking function causes an additional reduction of up to 20% of the total system overhead compared to push, for the Unbiased 0.2 and Unbiased 0.5 protocols. Figure 2(d) shows that ranking also reduces the mean provider-generator distance. Using HC alone results in the highest reduction in mean distance, but using the CF-HC combination results in less overhead and better yield compared to HC.

Using freshness adds overhead, increases the mean distance of the selected resource, and decreases yield. We attribute this to a local *flash crowd* effect. That is, all nodes within some local region assign similar high preference to the relatively few providers whose information is fresh. All nodes try to reserve resources on these few favored providers; clearly, not all can then be satisfied. Using CF or HC does not result in this effect, because CF and HC cause the formation of what we call local "node communities". When only CF is used, each query generator initially finds appropriate providers without any bias. Once CF values are learned, subsequent provider selection decisions are biased based on these CF values. Query generators then end up selecting the same provider that satisfied queries in the past. Each node learns different CF values based on its experiences with other nodes. This naturally distributes the preferences, avoiding flash crowding. The CF-HC combination avoids flash crowding most effectively. HC forces selection of nearby providers, keeping reservation requests and replies forwarding overhead down. At the same time, CF causes the formation of node communities. Thus, the CF-HC combination results in the smallest overhead.



(a) Percentage Query Satisfied

(b) Total System Overhead



(c) Yield

(d) Distance

Fig. 2. Dissemination rate = $(1/5)^{th}$ Query generation rate

B. Overlay Topologies

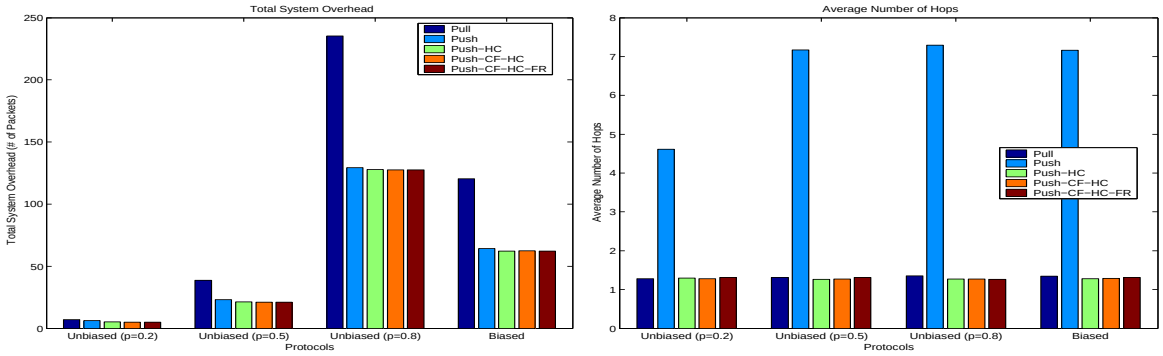
The proposed protocols and ranking functions can be viewed as orthogonal to the overlay topology of forwarding nodes. However, the overlay topology could have a significant influence on the effectiveness of the protocols, so we investigate several different topologies. In our experiments, we observed similar trends across tree, dense-random (average node degree 5.7) and sparse-random (average node degree 3.2) topologies. Due to space constraints, we are presenting those results in [12].

C. Effect of system load variation

Previous experiments show results for a *saturated* system load, wherein the average overall resource demand is equal to the available resources. In an *under-saturated* system, the number of resources exceeds demand. In an *over-saturated* system, resource demand exceeds supply. For the following results, we kept the information dissemination rate constant (equal to the query generation rate) and varied the system load to study its effect on the performance of ranking functions. To vary system load, we changed the initial resource availability on all nodes.

1) Under-saturated System: For an under-saturated system load, the resource units parameter is initialized to 50 on all nodes. For queries, resource units and resource time slot values were generated randomly between 1 and 5. Push protocols result in almost 100% query satisfaction, which is 1 or 2% higher than pull. At the same time, push overhead is reduced by 11% for Unbiased 0.2 protocol and by 46% for Biased protocol (Figure 3(a)). All ranking functions further reduce overhead. In particular, for Unbiased 0.2 protocol using push with CF-HC ranking function, resulting overhead is 16% less than pure push. Figure 3(b) shows that the mean provider-generator distance is also reduced by ranking functions, compared to pure push, where providers are selected randomly.

2) Over-saturated System: For an over-saturated system, the resource units parameter is initialized to 5 on all nodes. Here also, queries are generated by randomly selecting numbers between 1 and 5 as resource units and resource time slots parameter values. Figure 4(a) shows that the



(a) Total System Overhead

(b) Distance

Fig. 3. Under-saturated system, Dissemination rate = Query generation rate

push approach satisfies 2% to 6% more queries compared to pull. At the same time, push overhead is greater than pull overhead (Figure 4(b)). The CF-HC ranking function performed as well as pure push in terms of queries satisfied, but with less overhead compared to pure push. Figure 4(d) shows that all ranking functions outperform pure push in terms of mean provider-generator distance. In particular, HC beats CF-HC in this regard, but the CF-HC combination results in better yield (Figure 4(c)).

D. Comparison with pull caching

Caching responses for past queries can help populate information repositories and improve the performance of pull. In our implementation of "*pull-with-caching*", a query generator and all intermediate nodes on the overlay topology path traversed by the response, cache the response. The query generator first consults the local information repository populated by cached responses to find matching resources. Only when no matching resource is found, or reservation attempts on all matching resource providers are unsuccessful, are queries forwarded through the overlay topology in search of a required resource.

Compared to push, the pull approach with caching results in lower overhead but with a smaller percentage of queries satisfied. Whereas the information dissemination rate can be varied to bound overhead, in the pull caching case, the number of hops that the query is forwarded must be restricted. This restriction limits the number of nodes that are searched for matching resource, which in turn may reduce the query satisfaction ratio.

For each protocol, the "break even" information dissemination rate can be found, where the overhead of using push with ranking is equal to pull caching, but at the same time the number of queries satisfied is higher. Table I indicates the percentage query satisfied and the total system overhead (normalized with respect to pure pull overhead) of pull with caching and push with CF-HC combination at different frequencies for unbiased 0.2. For CF-HC(10)—an information dissemination rate of 1 every 10 cycles—the ranking parameters achieve the same performance as pull with caching. We found that for the Unbiased 0.5 protocol, the "break even" push rate is 15; that is, at an information dissemination rate of 1 every 15 cycles, the pull-ranking approach achieves the same performance as pull-caching with ranking. (Due to space constraints, these results are not included in a table or graph.)

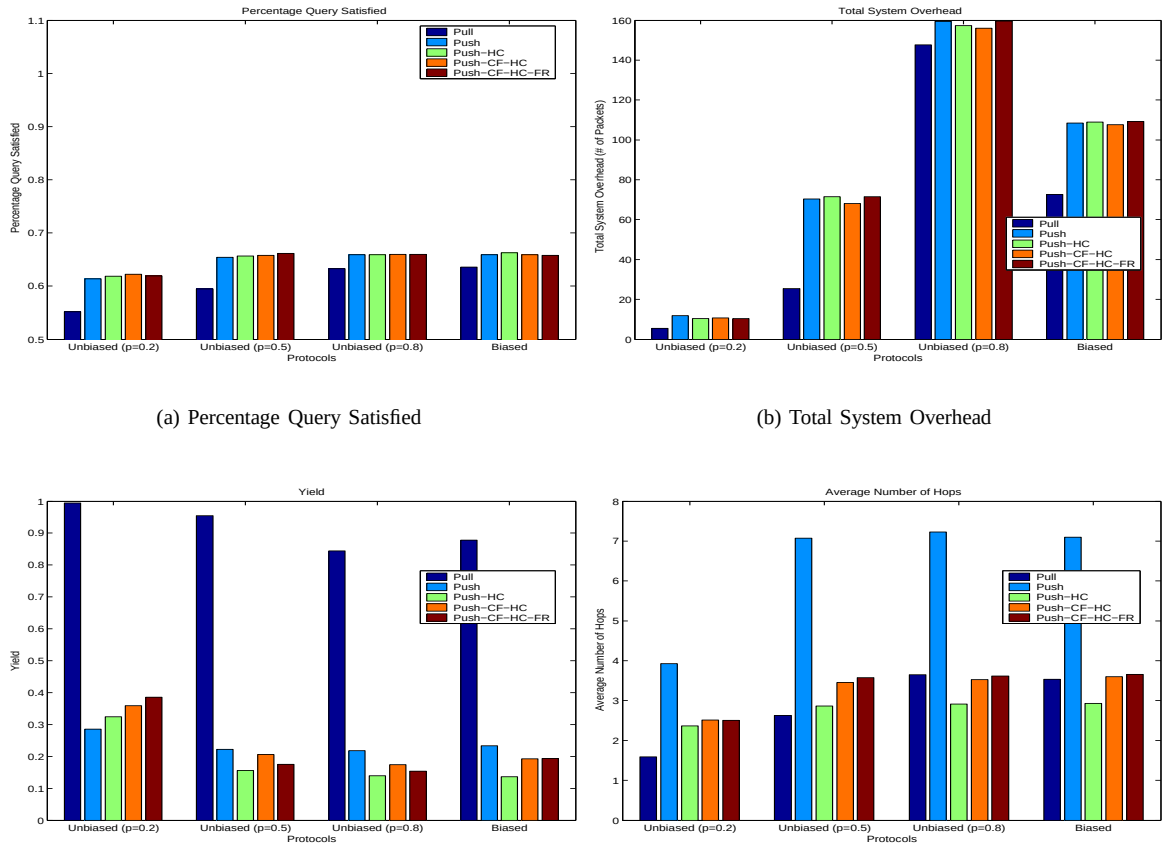
TABLE I. Comparison of pull-caching with push-ranking for unbiased 0.2

	Overhead	Percentage Query Satisfied
Pull	1.0000	82.03
Pull(caching)	0.3381	86.24
CF-HC(1)	1.1189	93.69
CF-HC(5)	0.4302	88.17
CF-HC(10)	0.3114	85.65
CF-HC(15)	0.2717	83.47
CF-HC(20)	0.2437	83.31

Also, while using information dissemination, control is distributed between providers and query generators. Providers decide how aggressively information should be disseminated and are able to adapt to low or high resource utilization by varying the information dissemination rate to attract more (or fewer) queries. In pull with caching, such distributed control cannot be achieved.

V. Conclusions and Future Work

In this paper, we investigate the effect of resource ranking policies on scheduler decisions. We evaluate the effectiveness of probabilistic push protocols across different push frequencies, offered application loads, and three different ranking combinations. Our results show that in general, the



(a) Percentage Query Satisfied

(b) Total System Overhead

(c) Yield

(d) Distance

Fig. 4. Oversaturated system: Dissemination rate = Query generation rate

combination of distance and past history leads to favorable schedules compared both with push and with the other two ranking combinations. We also show that including freshness in ranking can sometimes hinder performance. In future work, we plan to use the "feedback" received through application requests to effect the dissemination policy and ultimately to build adaptive dissemination protocols that react to the changing Grid conditions, thereby further helping schedulers make the most effective placement decisions.

References

[1] V. Iyengar, S. Tilak, N. B. Abu-Ghazaleh, and M. J. Lewis, "Nonuniform Information Dissemination for Dynamic Grid Resource Discovery," *Proc. of The 3rd IEEE International Symposium on Network Computing and Applications*, Boston, MA, August 2004.

[2] B. Gandhi, S. Tilak, M. J. Lewis, and N. B. Abu-Ghazaleh, "Controlling the Coverage of Grid Information Dissemination Protocols," *Proc. of The 4th IEEE International Symposium on Network Computing and Applications*, pp. 267–270, Boston, MA, August 2005.

[3] A. Iamnitchi, I. Foster, and D. Nurmi, "A peer-to-peer approach to resource location in grid environments," in *Symp. on High Performance Distributed Computing*, Aug. 2002. [Online]. Available: citeseer.ist.psu.edu/article/iamnitchi02peertopeer.html

[4] S. Tangpongprasit, T. Katagiri, H. Honda, and T. Yuba, "A time-to-live based reservation algorithm on fully decentralized resource discovery in grid computing." [Online]. Available: http://www.internetconference.org/ic2003/PDF/paper/tangpongprasit-sanya%.pdf

[5] A. R. Butt, R. Zhang, and Y. C. Hu, "A Self-Organizing Flock of Condors," *SC '03*, November 15-21, 2003, Phoenix, AZ.

[6] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *Proc. of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[7] K. Czajkowski, C. Kesselman, S. Fitzgerald, and I. T. Foster, "Grid information services for distributed resource sharing," in *10th IEEE International Symp. on High Performance Distributed Computing (HPDC-10)*, San Fransisco, CA, 2001, pp. 181–194.

[8] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 258–259, 2002.

[9] "The gnutella protocol specification v0.4," http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[10] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski, "Towards realistic million-node internet simulations," in *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, June 1999.

[11] E. Zegura and K. Calvert, "GT Internetwork Topology Models (GT-ITM)," http://www.cc.gatech.edu/projects/gtitm/.

[12] R. Desai, S. Tilak, B. Gandhi, M. Lewis, and N. Abu-Ghazaleh, "Analysis of Query Matching Criteria and Resource Monitoring Models for Grid Application Scheduling, Binghamton University, Comp. Science Tech. Report," http://grid.cs.binghamton.edu.