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-related state (e.g., load, availability, and location), and demand-related state (number and distribution of application resource requests) influence scheduling decision success. The scale of the Grid makes collecting and maintaining detailed up-to-date state information for all resources and requests impractical. Thus, concurrent distributed schedulers must make scheduling decisions based on incomplete resource state information. In this paper, we evaluate the effect that the criteria for selecting scheduling matches have 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 load models, and Grid overlay topologies. Among our findings is the counter-intuitive result that favoring freshness can sometimes harm 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 discover that pro-actively pushing dynamic state information to schedulers is beneficial.[1]

1. 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 effectively meet them. This problem is challenging because the environment is large scale, distributed, heterogeneous, and dynamic. Users submit many concurrent and possibly competing resource requests 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 of interest. This information can be made available using either the *push model* to periodically distribute summary 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. potentially inaccurate) information. On the other hand, the pull model requires significant delay at application run-time while the remote resource information is collected. 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. Section 4 provides more specific background and related work for Grid resource discovery and query matching.

Resource monitoring algorithms require a dissemination primitive to send resource information to schedulers (push model) or requests to resources (pull model). 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 [1], we studied the use of efficient probabilistic forwarding algorithms for disseminating resource information non-uniformly. 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 2. The first contribution of this paper is to analyze the effectiveness and overhead of different resource monitoring

1 This research is supported by AFRL contract FA8750-04-1-0054, NSF Award ACI-0133838, NSF Award CNS 0454298 and DOE Grant DE-FG02-02ER25526.

models (push, pull, and variants of each) using non-uniform dissemination protocols.

Resource monitoring algorithms make available to schedulers information regarding multiple matching resources. The second contribution of this paper is to explore the following questions: 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 a matching resource from the query generator (requester), the freshness of resource's information, the historical scheduling success to the resource, as well as combinations of these factors. The distance criterion helps in finding resources in the vicinity of a requester to reduce data transfer and application startup overhead. 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 3 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, pulling can improve schedules by freshening 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 5.

2. Resource Discovery Model

We consider a computational grid comprising a set of nodes organized (possibly self-organized) in an overlay topology. A node in our description corresponds to an aggregation of one or more resources, such as a Condor pool with a Centralized Manager, or a cluster with a designated Cluster Manager.

A 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). This abstraction of resource and query description can be extended to include more attribute-value pairs to accommodate multi-attribute queries. We model generic resources using type T. The discretized values of U enable a resource to be shared concurrently by multiple queries. The required time slots for a query (S) can be estimated by schedulers based on time requirements of similar queries as discussed elsewhere [2]. Note that U and S are described relative to a normalized standard and can be re-mapped locally to the existing hardware capabilities.

2.1. Architecture

The three main architectural components of our approach are the *Disseminator*, the *Query Resolver*, and the *Scheduler*: The Disseminator is responsible for disseminating and collecting resource information and queries. The Query Resolver performs query matching and resource ranking based on the information collected by the Disseminator, and determines the order in which the matching resources should be requested for a given query. Based on the ranks assigned by the Query Resolver, the Scheduler contacts the corresponding resources to schedule a given query. All nodes in the topology run these three components, which we describe in more detail below.

2.1.1. Disseminator: The Disseminator encapsulates the dissemination function needed by both the push and the pull models to distribute resource information (push) or resource queries (pull) to other nodes in the network. We evaluate the Biased and Unbiased protocols developed in our earlier work [1] as dissemination primitives. These protocols use a probabilistic approach to disseminate information more frequently to nearby nodes than to remote nodes; thus, the protocols capitalize on the intuition that most queries are best scheduled on nearby resources. 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 equivalent to gossipping: intermediate nodes forward the information with a constant probability P. We refer interested readers to [1, 3] for more details. This previous work characterizes the dissemination process but does not address query matching, which we explore in this paper.

2.1.2. Query Resolver (QR): When a query cannot be satisfied locally, the Query Resolver consults its local repository. In the push model, dissemination leads to knowledge of multiple resources that can support a query. The QR matches the generated query's attributes 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. We consider the following ranking criteria:

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.

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 requester's confidence in a provider node increases when the provider accepts the request, and decreases when it rejects the request.

We use these ranking criteria 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 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. Because of the normalization, the final rank ranges between 0 and 1. This ranking formula can be instantiated with different combinations of W_i, we present results for a handful of interesting possibilities.

In general, the rank of a resource provider can be a complex value that is multi-dimensional and application-dependant. Although, we currently compute rank using a linear function of ranking heuristics, our scheme can be easily extended to more complex non linear functions.

2.1.3. Scheduler: In the pull model, the scheduler receives queries generated at other nodes and matches these queries against its own node descriptor tuple. If query matching is successful, it sends back a response to the query generator. The scheduler (running on $node_A$) uses the candidate resource list prepared by QR (push model) or query responses in the order they are received (pull model). The scheduler then sends a resource request to the scheduler running on the provider node ($node_B$). The scheduler on $node_B$ matches request against its own node descriptor tuple. This matching is required because $node_A$ may have used stale information and resource state on $node_B$ might have changed. If matching is successful, the scheduler on $node_B$ can accommodate the given request. In this case, it reduces available slots and sends back a *request accepted* reply to $node_A$; otherwise it sends back a *request rejected* reply. Upon receiving a request rejected reply, the scheduler on $node_A$ removes $node_B$ from its candidate resource list and proceeds to consider the remaining candidate nodes. Upon receiving a request rejected message from all the candidate nodes, the scheduler can resubmit the request at a later time. However, we do not enforce a policy regarding request re-submission at this time.

3. Experimental Evaluation

In this section, we compare non-uniform pull with non-uniform push and study effect of ranking on non-uniform push using simulations. We use the Scalable Simulation Framework Network (SSFNet) for all experiments.

We evaluate the performance of our ranking scheme across non-uniform protocols [1], specifically Unbiased (with probabilities 0.2, 0.5 and 0.8) and Biased protocols. The selected probabilities for the unbiased protocol are representative of low, moderate and high network coverage [3]. The biased protocol results in very high coverage near the source and low coverage further away. Nodes disseminate queries (pull) and information (push) using non-uniform protocols. The following four performance metrics characterize the effectiveness of the schedule generated using the dissemination protocols: percentage of queries satisfied, packet overhead, distance of the selected node from the requester, and yield. Five types of messages—namely, information, query, query response, request, and request reply—are forwarded through the overlay topology using shortest path routes. On each node, packet overhead is calculated in terms of the total number of messages forwarded during the simulation. Yield is the ratio of satisfied query per request.

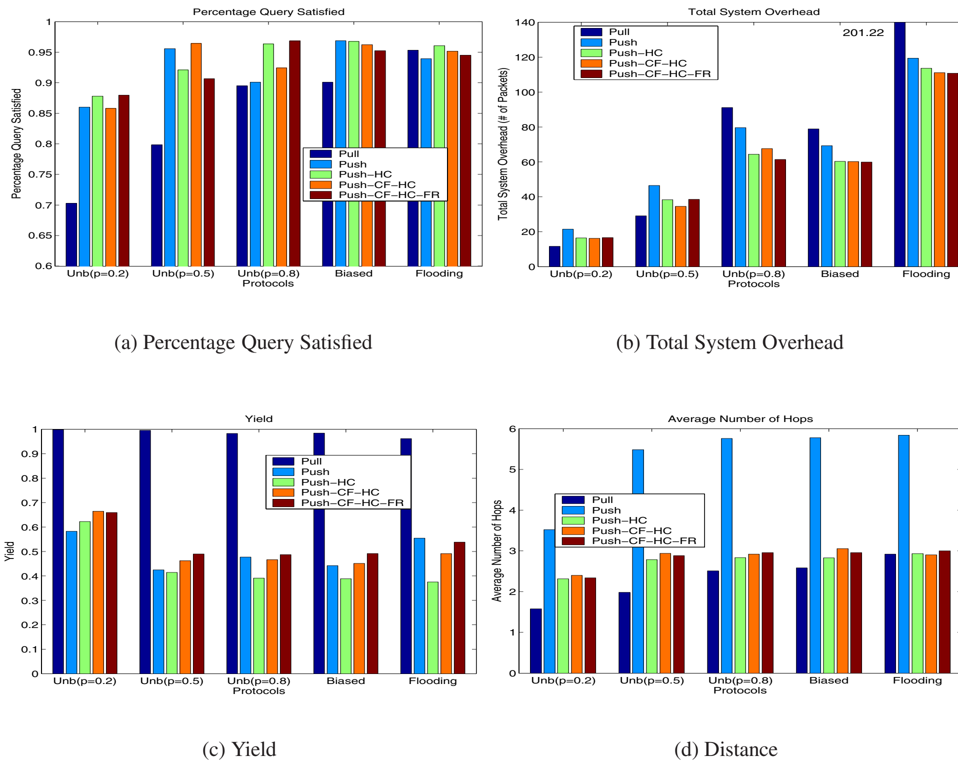
The confidence factor (CF) value is constrained to the range $[0, 1]$, and begins at a neutral value of 0.5. We use a simple feedback control scheme to adjust CF. When a provider node accepts a request, the requester's confidence in that provider is increased by 0.05; when a provider rejects a request, confidence is decreased by 0.05. We measure information freshness in terms of simulation cycles rather than simulation clock seconds. Therefore, we consider all information received within the same simulation cycle equally fresh. When we consider multiple criteria in the ranking formula, each of them is assigned equal weight.

In the following experiments, we consider only dedicated Grid resources that are available throughout the simulation, not resources that join or leave the Grid dynamically. We run simulations for 100 simulation cycles. So, the initial resource time slot value is set to 1000 on all providers. Also, all nodes provide resources with 10 initial resource units.

3.1. Study of Non-uniform Protocols

Figure 1 shows simulation results for a tree topology of 100 nodes. In this simulation, each node generates one query in each simulation cycle. In the push model, each node also disseminates resource information in each cycle. That is, the information dissemination rate is equal to the query generation rate. Our observations are:

- With the increase in forwarding probability (for Unbiased), more nodes receive queries (pull) or information (push). Therefore, more queries are satisfied and



(a) Percentage Query Satisfied

(b) Total System Overhead

(c) Yield

(d) Distance

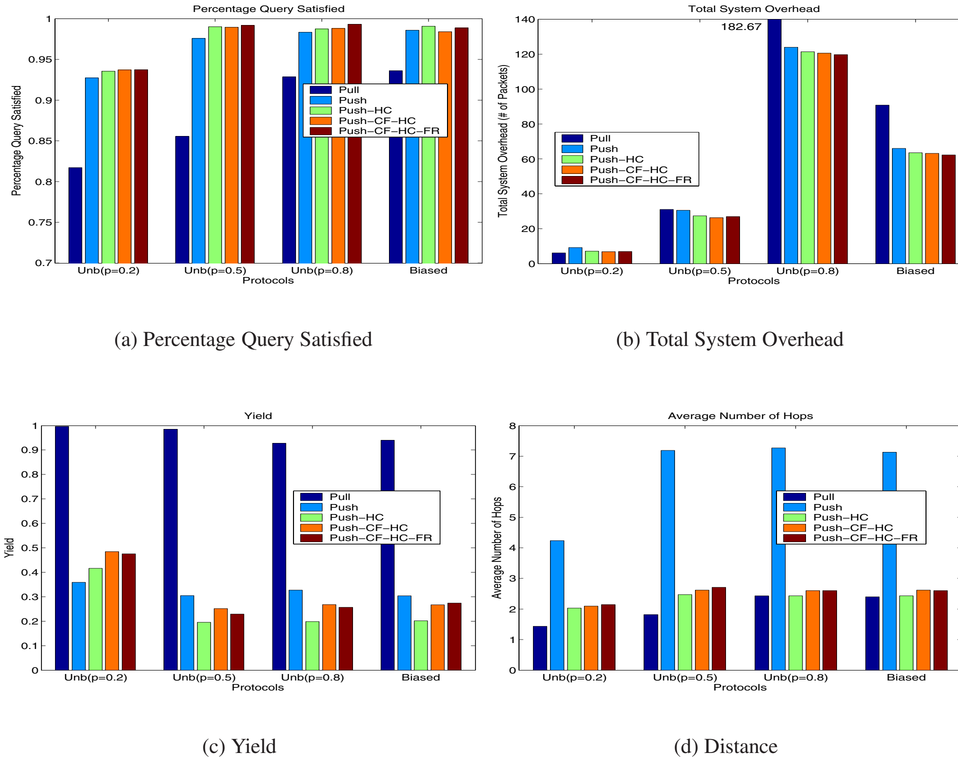
Figure 1. Tree Topology: Dissemination rate = Query generation rate

more overhead is incurred. Using non-uniform protocols with ranking (e.g. Unbiased 0.8, Biased), percentage queries satisfied approaches that of flooding at significantly less overhead (Figure 1(a) and 1(b)).

- Push satisfies more queries than pull across the board. Specifically, for Unbiased 0.2 push satisfies 16% more queries (Figure 1(a)) than pull. The reason is that in push, a requester uses information collected over many previous cycles whereas in pull, a query reaches out to only a limited set of providers.
- For low probabilities in Unbiased (0.2 and 0.5), the dissemination (of information or query) is localized. Thus, even though push and pull have equal dissemination overhead, total overhead of push is more than pull in these two cases. This is because push satisfies more queries than pull and therefore, more request and request reply messages are processed. However, when most of the nodes receive disseminated queries (in Unbiased 0.8 pull, Biased, and flooding), all of the nodes that can satisfy the query return query response. Therefore, at higher probabilities, pull overhead surpasses push overhead.
- Pull always outperforms push in terms of yield (Figure 1(c)). In the pull model, a requester sends request to a potential provider as soon as it receives first query response. Because a potential provider generates a response after considering up-to-date resource state information, the probability of a request being accepted is high. This results in high yield for pull. For push, the QR uses all the available information in the local repository—including potentially stale information—for preparing the candidate resource list for the scheduler. The scheduler utilizes this list and sends multiple requests for each query, thereby resulting in low yield.
- Also, in pull, a requester receives first query response from a nearby provider. Therefore, it is highly likely that queries will be satisfied locally (at small distance). For push without ranking, the QR adds matching resources in the candidate list in random order, resulting in possibly far away nodes being selected by the scheduler (Figure 1(d)).

3.2. Study of Ranking parameters

Figure 2 shows the performance results for a 600-node transit-stub type random topology. The information dissemination rate is equal to the query generation rate. Our observations for this topology are in line with tree topology's results. For example, Figure 2(a) shows that for Unbiased 0.2, push increases query satisfaction by 11% compared to pull.



(a) Percentage Query Satisfied

(b) Total System Overhead

(c) Yield

(d) Distance

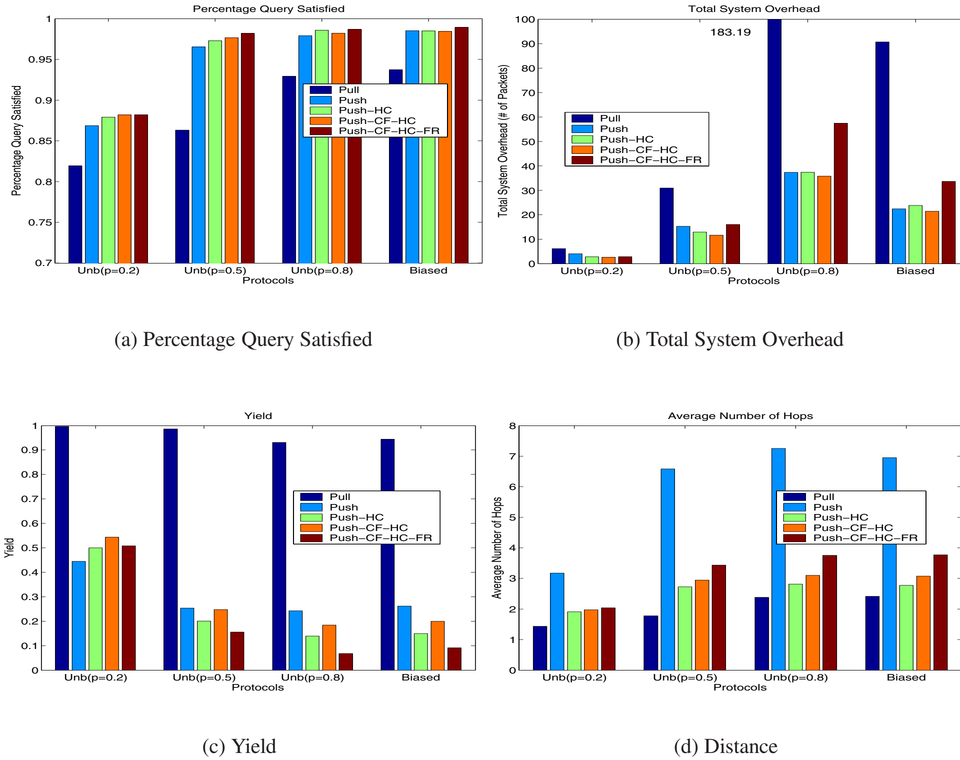
Figure 2. Dissemination rate = Query generation rate

And, up to 32% reduction in overhead is achieved in the case of Unbiased 0.8 (Figure 2(b)).

Ranking functions tend to improve the query satisfaction ratio and reduce overhead. We experimented with all possible ranking parameter combinations. We present results only for interesting combinations. The hop count (HC) component of ranking functions favors nearby resource providers, which results in selecting nearby provider nodes and hence less overhead. For example, Figure 2(b) shows that for Unbiased 0.5, push with a CF-HC ranking function reduces overhead by up to 14% compared to pure push. Figure 2(d) shows that using ranking functions helps find resources in the vicinity of query generators.

During simulation, all nodes adjust their CF values to reflect their experiences with other provider nodes. After some time, the CF component starts dominating the HC component when used together. This domination of CF results in better yield for CF-HC combination compared to only HC. Figure 2(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. When ranking is used, top ranked resources are highly likely to accept a request, resulting in better yield compared to pure push. Figure 2(c) shows that for Unbiased 0.2, yield is improved when ranking is used compared to pure push. However, for Unbiased 0.5, 0.8, and Biased protocols, pure push results in better yield compared to push-ranking. Because of high forwarding probability in these protocols and higher network connectivity, nodes have information about more resources. We conjecture that push with random resource selection outperforms push-ranking in such cases, because all ranking functions considered here cause aggressive search in the vicinity of resource requesters.

To further study the effectiveness of ranking , we reduce the frequency of information dissemination. Figure 3 shows results for the case when the information dissemination rate is $(1/5)^{th}$ of the query generation rate. The 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. We note that push results in higher percentages of queries satisfied than pull. For Unbiased 0.5 push increases the percentage of queries satisfied by 10% (Figure 3(a)). Furthermore, push overhead is reduced even more. Figure 3(b) shows that even for Unbiased 0.2 push overhead is less than pull overhead. Overall, push overhead is 35% to 80% less than pull overhead. Using any ranking function performs as well as pure push in terms of the percentage of queries satisfied (Figure 3(a)). Figure 3(b) shows that the CF-HC ranking function causes



(a) Percentage Query Satisfied

(b) Total System Overhead

(c) Yield

(d) Distance

Figure 3. Dissemination rate = $(1/5)^{th}$ Query generation rate

an additional reduction of up to 20% of the overhead compared to push, for the Unbiased 0.2 and Unbiased 0.5 protocols. Figure 3(d) shows that ranking also reduces the mean provider-requester 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 than 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 secure 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 overhead down. At the same time, CF causes the formation of node communities. Thus, the CF-HC combination results in the smallest overhead.

3.3. 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* sys-

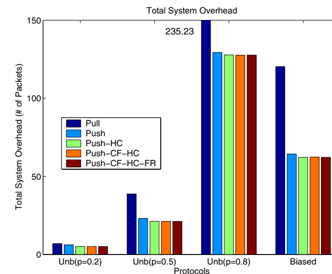


Figure 4. Under-saturated system, Total System Overhead

tem, 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. We present results for random topology of 600 nodes.

3.3.1. 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 4). All ranking functions further reduce overhead. In particular, for the Unbiased 0.2 protocol using push with CF-HC ranking function, the resulting overhead is 16% less than pure push.

3.3.2. 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.The push approach satisfies 2% to 6% more queries compared to pull. At the same time, push overhead is greater than pull overhead (Figure 5(b)). This is because when the system used the pull approach, few nodes return query responses upon receiving queries, due to overloading. For push, requesters must make more attempts to secure resource on providers. When the system is overloaded, many requesters are competing for less resources, causing rapid resource state changes. These rapid changes make resource information used by providers stale. The CF-HC ranking function performs as well as pure push in terms of queries satisfied, but with less overhead compared to pure push.

3.4. 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-caching*", a requester and all intermediate nodes on the overlay topology path traversed by the response, cache the response. A requester first consults cached information, and only once all attempts to secure the resource using cached information fail, the query is forwarded.

Compared to push-ranking, the pull-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 resources, 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-ranking is equal to pull-caching, but at the same time the number of queries satisfied is higher. Table 1 indicates the percentage of queries satisfied and the total system overhead (normalized with respect to pure pull overhead) of pull-caching and push with the CF-HC combination at different frequencies for unbiased 0.2. For CF-HC(10)—an information dissemination rate of once every 10 cycles—the ranking parameters achieve the same performance as pull-caching. We found that for the Unbiased 0.5 protocol, the "break even" push rate is 15. 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.

	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

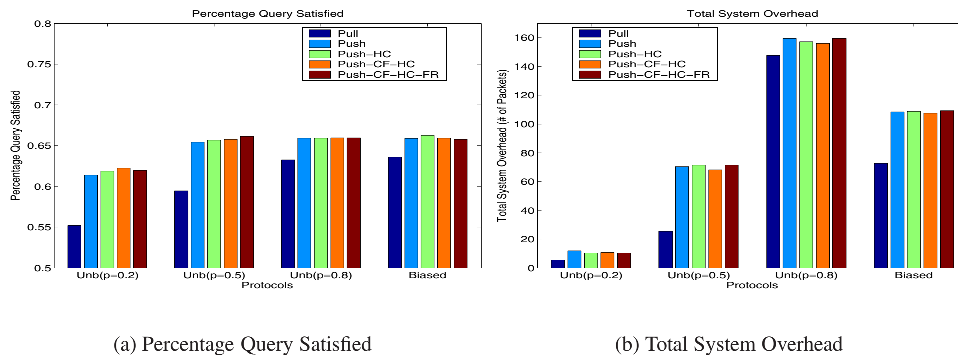
Table 1. Comparison of pull-caching with push-ranking for unbiased 0.2

4. Related Work

Iamnitchi et.al. [4] proposed pull strategies for query processing and evaluated scheduling success using average hop distance travelled by a query. Our study includes this metric, in addition to the percentage of queries satisfied, and the potential overhead of resource selection. We encapsulate their "best neighbor" heuristic in our *confidence factor* ranking component and consider the effect of two additional heuristics in ranking—higher preference to nearby resources, and 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 [5] 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. Furthermore, in this model a resource is reserved exclusively for one query, whereas in our model, a resource can be concurrently shared by multiple requests.

In the Flock-of-Condors approach [6], *Condor pools*–organized in a P2P structure using the Pastry routing



(a) Percentage Query Satisfied

(b) Total System Overhead

Figure 5. Over-saturated system: Dissemination rate = Query generation rate

protocol–disseminate resource information and sharing policies (collectively called ClassAds [7]) to neighbors. Pools contact one another to negotiate resource sharing. Thus, this approach uses a combination of push and pull. The authors study 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.

Grid Information Services [8] require resources to be registered with the MDS directory servers. Resources use a soft-state protocol to periodically update their information in directory servers. Clients use an enquiry protocol to obtain information about current resource status and sharing policy from directory servers. Resource monitoring and discovery services use the directory service, however Directory servers' organization, policies for information dissemination, and resource selection criteria are left unspecified.

The *Network Weather Service (NWS)* [9] is a resource performance forecasting system. It uses time-series data about past resource performance to statistically derive a prediction of resource status in next time interval. In our model, prediction data can be disseminated to enable schedulers in making intelligent placement decisions.

SWORD [10] is scalable wide-area resource discovery system. Although, it strives to provide similar functionality, there are significant architectural differences. While SWORD uses DHT over a structured overlay, we assume an unstructured network and use gossiping protocols.

5. Conclusions and Future Work

In this paper, we investigate the effect of resource tracking models and resource ranking policies on scheduler decisions. We use probabilistic protocols for proactive and reactive resource tracking and evaluate the effectiveness of three different ranking criteria and their combinations on the quality of schedules across different topologies, push frequencies, and offered application loads. Our results show that in general, the 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 result in a "local flash crowd" and harm overall 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," *NCA*, 2004.

[2] J. Cao, S. Jarvis, S. Saini, D. Kerbyson, and G. Nudd, "ARMS: An agent-based resource management system for grid computing," *Scientific Programming*, 2002.

[3] B. Gandhi, S. Tilak, M. J. Lewis, and N. B. Abu-Ghazaleh, "Controlling the Coverage of Grid Information Dissemination Protocols," *NCA*, 2005.

[4] A. Iamnitchi, I. Foster, and D. Nurmi, "A peer-to-peer approach to resource location in grid environments," in *HPDC*, 2002.

[5] S. Tangpongprasit, T. Katagiri, H. Honda, and T. Yuba, "A time-to-live based reservation algorithm on fully decentralized resource discovery in grid computing."

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

[7] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *HPDC*, 1998.

[8] K. Czajkowski, C. Kesselman, S. Fitzgerald, and I. T. Foster, "Grid information services for distributed resource sharing," in *HPDC-10*, 2001.

[9] R. Wolski, N. T. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems*, vol. 15, no. 5–6, pp. 757–768, 1999.

[10] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and Implementation Tradeoffs for Wide-Area Resource Discovery," *HPDC-14*, July 2005.