Appendices
APPENDIX - A

Creating a Grid Resource in GridSim

A Grid Resource simulated in GridSim contains one or more machines. Similarly a machine contains one or more PEs (Processing Elements or CPUs). Below are the steps to be followed in JAVA code to create a Grid Resource.

1. Create an object of type MachineList to store one or more Machines

   MachineList mList = new MachineList();

2. A Machine contains one or more PEs/ CPUs. So we create an object of type PEList to store this PEs before creating a Machine.

   PEList peList1 = new PEList();

3. Create PEs and add these into the object of PEList created in step 2. We have to specify the unique ID of the PE as first parameter and its MIPS (Millions of Instruction per Second) rating as second parameter.

   peList1.add( new PE(0, 377) );

4. Create a Machine with its unique ID and the PEList asscociated with it.

   mList.add( new Machine(0, peList1) );

5. Repeat the steps from step 2 to step 4 to create additional number of machines.

6. Create a Resource Characteristics object which will store the properties of a Grid Resource, its architecture, Operating System, List of Machines, Allocation Policy which can be Time Shared or Space Shared, Time Zone and Cost of the Resource in terms of $ per PE Time Unit.

   ResourceCharacteristics resConfig = new ResourceCharacteristics(
       arch, os, mList, ResourceCharacteristics.TIME_SHARED,
       time_zone, cost);
7. In the final step we create an object of Grid Resource specifying its name, communication speed, peak load, off-peak load, holiday load and list of holidays along with the Resource Characteristics object which was created in the step 6.

\[
\text{GridResource gridRes= new GridResource(name, baud\_rate, seed, resConfig, peakLoad, offPeakLoad, holidayLoad, Weekends,Holidays);}\
\]
APPENDIX – B

Enclosed Reprint of Published Papers


EFFICIENT TASK-SCHEDULING IN GRID COMPUTING USING PRIORITIZED SSL WITH BACKEND FORWARDING MODEL

G. THILIPKUMAR¹  Dr. R. BALASUBRAMANIAN²

¹Ph.D., Research Scholar, Department of Computer Science, J.J. College of Arts and Science, Pudukkottai.
²Professor, Department of Computer Science, J.J. College of Arts and Science, Pudukkottai.

ABSTRACT

Over the years, grid computing has emerged as one of the most viable and scalable alternatives to high performance supercomputing, tapping into computing power of the order of Gigaflops. However, the inherent dynamicity in grid computing has made it extremely difficult to come up with near-optimal solutions to efficiently schedule tasks in grids. The present paper proposes a novel grid-scheduling heuristic that adaptively and dynamically schedules tasks without requiring any prior information on the workload of incoming tasks. The approach models the grid system in the form of a state-transition diagram, employing a prioritized round-robin algorithm with task replication to optimally schedule tasks, using prediction information on processor utilization of individual nodes. Simulations, comparing the proposed approach with the round robin heuristic, have shown the given heuristic to be more effective in scheduling tasks as compared to the latter.

Keywords: Dynamic Scheduling, Grid Computing, Task Replica, Round Robin, Prioritized Round Robin, Prediction Information.

1. Introduction

Grid computing is a form of distributed computing, where a set of loosely coupled and heterogeneous computing nodes donate their unused processor cycles to create a pool of substantial processing capacity. In recent years, grid computing has emerged as one of the most feasible alternatives to process compute-intensive tasks, using co-operating processing nodes of ordinary capacity. The main advantage that grid computing offers, is that inexpensive computing nodes are coupled together to produce resource capacity comparable to high-end supercomputers, albeit at a lower cost. The major bottleneck that grids face is that individual processors might not be well connected to one another and thus, the model is more suited to applications which can be broken into several independent and atomic sub-tasks, without any requirement to communicate intermittent results between the grid nodes.

The principal challenge involved in any distributed computing environment, is that optimal scheduling of tasks dynamically entering the grid, becomes an NP-hard problem. Efficient grid scheduling is one of the key factors for achieving high performance in grid environments. Several heuristics [1, 2, 3, 4, 5] have been proposed in literature to schedule the tasks efficiently to the most suitable node present in the grid. A majority of these heuristics show satisfactory results in static grid environments. However, they cannot be directly applied in dynamic environments where tasks are continuously arriving in the grid at regular intervals.

The principal motivation of the present paper is to develop a scalable task-scheduling algorithm which can operate efficiently without the services of a full-fledged prediction system providing prior information on workload of incoming tasks. The paper proposes an
enhancement of the existing round-robin heuristic [2], where we exploit information on the capacity of individual grid nodes, to prioritize tasks currently in execution, such that tasks currently allocated to slower machines are preferred for replication purpose over jobs executing on comparatively faster machines. The approach facilitates replication of tasks, hitherto assigned to execute on slower machines, on machines with higher processing capacity.

2. Related Work

There has been significant research in the past to study the classic problem of optimal job assignment in distributed environments such as grids. Heuristics, dedicated to scheduling tasks optimally in a grid environment, can be broadly categorized into the following classes.

A. Batch Mode Mapping Heuristics

Tasks are queued and collected into a set when they arrive in the batch mode [1]. They will be scheduled or mapped to their respective machines afterwards at a specific interval by the scheduling algorithm. In Min-min scheduling algorithm, each job will be always assigned to the resource which can complete it the earliest in order to spend less time completing all jobs. The Max-min scheduling algorithm is similar to Min-min scheduling algorithm except that it gives the highest priority to the job with the maximum earliest completion time. The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would “suffer” most in terms of expected completion time if that particular machine is not assigned to it.

B. On-line Mode Mapping Heuristics

Tasks are scheduled as soon as they arrive [1]. All the machines in the grid need to be referred to in order to decide the optimal machine that needs to be mapped to the incoming task. The MCT (minimum completion time) heuristic assigns each task to the machine that results in the task’s earliest completion time. The MET (minimum execution time) heuristic assigns each task to the machine that performs the task’s computation in the least amount of execution time. The mapping algorithm takes O(m) time (where m is the number of machines in the grid) to map a task to its optimal machine. If a task arrives in the Grid while the mapping algorithm is underway, it has to wait until the previous mapping is complete.

C. Ringed Round Robin Algorithm

The algorithm assumes no prediction information is available either for the task length or the processor capability [2]. Tasks arriving in the Grid are assigned to idle machines in a FCFS (First come First serve) manner. Task replication is performed later on, in a ringed round robin fashion, to achieve better resource utilization in the Grid. Both the task allocation and task replication routines are executed randomly without any basis or optimization criteria.

D. Scheduling Algorithm for Bag-Of-Tasks using Multiple Queues with duplication

The strategy of the algorithm is to schedule tasks according to their workloads and computing power of resources available [6]. The machines in the Grid are ranked according to their processing speed and are allocated tasks that are most compatible to their processing capacity. In addition, it adopts a duplication scheme in order to achieve optimal utilization of the machines available and to avoid undesirable scheduling decisions. The algorithm is static in nature as compared to the inherent dynamic nature of grid systems where tasks are continuously submitted to the system and get optimally scheduled on the machines available in the grid.

E. Self-Adaptive Scheduling System for Heterogeneous Computing

Ming Wu and Xian-He Sun [7] propose a prediction model based on probabilistic
approach for long-term, application-level based task scheduling system in a distributed heterogeneous computing environment. The architecture of the distributed model comprises of a task allocator, scheduler and predictor, all integrated together, to form the GHS scheduling system. A self-adaptive task scheduling algorithm, based on probabilistic approach, is put forth to improve the accuracy of the GHS scheduling system for efficiently scheduling the meta-tasks.

F. Economic Task Replication for Scheduling in Distributed Systems

Amit Agarwal and Padam Kumar [8] attempt to minimize the number of task replications without affecting the overall make span of the meta-task submitted to the grid, by proposing two workflow scheduling algorithms, namely, Reduced Duplication for homogeneous systems (RD) and Heterogeneous Economical Duplication (HED) for heterogeneous systems respectively. The proposed algorithms aim at optimizing the overall processor consumption, by removing some duplicated tasks in the schedule whose removal does not affect the make span adversely, thereby producing scheduling holes in the system, which can, in turn, be used to schedule other distributed applications in the grid.

G. Task Duplication based Scalable Scheduling Algorithm for Symmetric Multiprocessors

Oh-Han Kang and Dharma P. Agrawal [11] present a task duplication based scalable scheduling algorithm for Symmetric Multiprocessors (SMP), referred to as the S3MP (Scalable Scheduling for SMP), to schedule the tasks of a DAG onto a bus-based SMP environment, facilitating duplication of certain critical tasks, so as to reduce the overall schedule length, by pre-allocating communication resources so as to avoid communication conflicts later on during scheduling.

H. Miscellaneous Heuristics

Bin Zeng et al. [3] propose a negotiation based model, where adaptive learning agents, representing individual resources and tasks, cooperate among themselves to help achieve a nearoptimal schedule. N.Malarvizhi and V.Rhymend Uthariaraj [4] describe a scalable grid-architecture involving a Grid Resource Manager, assuming the role of a resource broker to select computational resources based on job requirements and the capacity of grid resources, so as to minimize the time to process each application along with transmission time associated with it. D. P. Spooner et al.[5] develop a multi-tiered scheduling architecture (TITAN) that uses a performance prediction system (PACE), along with brokers that are involved in distribution of jobs in the grid, to meet deadlines and significantly increase the efficiency of resource utilization. The paper [9] presents a novel load balancing approach in a heterogeneous distributed environment. The scheduler takes into account the threshold value, based on the ratio of service rates, along with the queue length to determine whether it is beneficial to migrate a given local task to another node in the system or not. Markov process model is used to describe the behavior of the heterogeneous distributed system under the proposed policies. Ruay-Shiung Chang et al. [10] propose an Adaptive Scoring Job Scheduling algorithm (ASJS) for a distributed grid environment to reduce the completion time of submitted jobs, by assigning jobs to resources after looking into recent scheduling history of every available resource and then choosing the most optimal one. Computing intensive jobs and data intensive jobs are handled differently, and local and global updates are used to obtain the most recent status of grid resources to schedule jobs more effectively in real time. System ModelSyed Nasir Mehmood Shah et al. [12] propose an algorithm for CPU scheduling of a modern multiprogramming operating system, design and development of new CPU scheduling algorithms (the Hybrid Scheduling Algorithm and the Dual Queue Scheduling Algorithm) with
a view to minimize overall task schedule. The following paper extends this prioritized round robin heuristic from a single system multiprogramming environment, onto a multi-processor distributed architecture.

3. System Model

We consider a Grid system comprising of distributed computing nodes, and a central server for task allocation purpose. The Grid is modeled in the following state transition diagram, to be oscillating between the 4 given states. The system comprises of two queues to store records of the tasks currently in the Grid, namely the Waiting Queue and the Execution Queue. The Waiting Queue comprises of tasks in the Grid which are yet to be mapped to their respective machines, while the Execution Queue contains all the tasks which are currently in execution on at least one of the machines in the Grid.

The proposed model works under the following assumptions:

- The model assumes that the tasks arriving in the Grid are atomic (cannot be broken into further sub-tasks) and are independent of one another.

- It is assumed that the transportation costs involved, when tasks are mapped to their respective machines, are considered to be negligible.

- It is assumed that we have no information available on the workload of the incoming tasks as it is not practically feasible to derive information regarding the same without the services of a full-fledged Prediction System.

- The approach assumes that the processing speed of individual computing nodes is available to us. The initial processing speed of nodes is provided and the processing capacity of machines is then updated from time to time on the basis of the last application executed (workload) and the time taken.

4. Proposed Solution

We represent the heuristic as a state transition diagram (Given at fig-1.) with the grid occupying one of the four states at any given time. The waiting queue, comprising of tasks waiting to be mapped and executed on their respective machines, is implemented as a First in, First out (FIFO) queue where the task with the earliest arrival time is at the head of the queue and allocated an idle machine before other tasks waiting in the queue. The execution queue, consisting of tasks currently in execution, is implemented as a circular queue where each task in the queue has a specific order, and no task has the same order as any other task. The Execution Queue makes use of three pointers to scan the circular list in a Ringed Round Robin fashion.

The current pointer is used to point to the task currently having the highest priority in the Execution Queue. The next pointer is used to point to the task with the second highest priority, which is nothing but the task lined next to the current task, one step in the clockwise direction. The last pointer is used to point to the task with the least priority in the Execution Queue, which is precisely the task placed besides the current task, a step in the anti-clockwise direction. The following is the description of the four states occupied by the grid system, during the course of time.
State I

Initial phase: State I is represented by an idle scheduler waiting for tasks to arrive in the grid. Incoming tasks are lined up in the waiting queue. Both the Waiting Queue and the Execution are initially empty. When the number of tasks in the Waiting Queue becomes more than the threshold value, a transition is made to State II.

State II

Initial phase: The Execution Queue is initially empty while the Waiting Queue comprises of a number of incoming tasks in the Grid. We maintain a list comprising of idle machines in the system. Initially all the machines are idle in State II, and hence the list contains all the machines in the Grid.

The tasks from the head of the Waiting queue and mapped one by one to the machines in the idle list. As soon as a task from the Waiting queue gets mapped to a machine, the given machine is subsequently removed from the idle list, while the task is removed from the head of the Waiting Queue and inserted into the Execution Queue as explained in the System Model. The logic behind the mode of insertion is to give the highest priority to a task which has been assigned the slowest processor and vice-versa. The task with the highest priority in the Execution Queue would be the first one to get replicated because it is essentially the task with the slowest processors dedicated to it and hence replicating such a task would lead to a very high probability of the new machine executing the task before the machines already assigned to it.

State III

Initial phase: The waiting Queue is initially empty while the Execution Queue consists of tasks that are in execution in the Grid.

If a machine completes the execution of a task, the processor list of that particular task is referred to, and all the machines dedicated to executing the given task are released and made free while the task is removed the Execution Queue and the current, next and last pointers are updated if required. We update the processing power of the newly freed machine based upon the number of instructions that the machine executed for the previous task and the time it took to finish its execution. If the processing speed of the machine is greater than that of the maxProcSpeed of the task pointed to by the current pointer, then the given machine is required to execute the replica of the task pointed to by the current pointer.

Also, the current, next and last pointers are updated as follows. The current pointer becomes the last pointer, the next pointer becomes the current pointer and the next pointer would now be pointing to the task that was one step in the clockwise direction to the task pointed to by the erstwhile next pointer. Also, if the machine assigned to execute the replica of a task has a processor speed greater than that of maxProcSpeed of that task then the value needs to be updated to the processing speed of the given machine. We also keep track of the number of machines executing replicas and the number of tasks in the Waiting Queue, the information of which is exploited in State IV of the heuristic. At this point of time, we also check if there are any other idle machines present in the Grid from the idle list and assign tasks from the Execution Queue in the same way as described above, one by one, to these idle machines which are then subsequently removed from the idle list.

However, if a machine is found to have its updated processing speed to be less than that of the maxProcSpeed of the current task, then we do not assign the machine to execute the task replica for the simple reason that in all probability, the task would be accomplished faster on one of the machines already assigned to it as compared to the given machine. The machine is then inserted at the tail of the list containing the idle machines present in the Grid.
There are three scenarios, eventually possible, in State III.

Case I: All the tasks in the Execution Queue are successfully completed while the Waiting Queue is still empty. In this case, we traverse back to State I.

Case II: All the tasks in the Execution Queue are successfully completed while the number of tasks in the Waiting Queue is still less or equal to the threshold. In this case, we traverse back to State II.

Case III: The number of tasks in the Waiting Queue has exceeded the threshold before all the tasks in the Execution Queue could be completed. In this case, we traverse to State IV.

State IV

Initial Phase: Both the Waiting and the Execution Queue are initially non-empty. Two scenarios are possible at this point of time.

Case I: The number of machines executing replicas is less than the number of tasks in the Waiting Queue.

Case II: The number of machines executing task replicas is more than the number of tasks in the Waiting Queue.

The tasks in the Execution Queue are traversed in an anticlockwise manner one by one, starting from the task pointed to by the last pointer (the least priority task) and if a task has more than one machine allocated to it, the machine at the tail end of the processor list is taken out of the list, freed from the task it was currently executing and assigned the task at the head of the Waiting Queue (after removing the task from the queue).

In Case I, we stop the traversal as soon as all the tasks in the execution queue are being run on one and only one machine and transition to State II. In Case II, we stop traversing the linked list as soon as all the machines in the Waiting Queue are assigned a machine, and then subsequently transition to State III.

5. Simulation

Simulation consists of a grid network comprising of a fixed number of computing nodes. Both, the number of machines as well as their processing capacity, are chosen randomly over a pre-defined range. Tasks enter the grid at random intervals within the range of 10 units and are assigned to the scheduler for allocation as soon as they arrive.

The simulation code defines one time unit as a single iteration, where we check for the current state of the grid system and accordingly perform the amount of work the scheduler can do in a single time unit when the grid is in that particular state. We run the simulation for as many steps as required to complete all the jobs submitted to the grid over the span of time. For our simulation purpose, we have considered the four test cases based on the degree of heterogeneity in node capacity and task workload entering the grid. In each case, we have considered 5 input cases with the following number of tasks-40, 80, 120, 160, 200. The following the four test cases in consideration.

We have considered the overall turnover time as our yardstick to test the performance of the proposed heuristic against the round-robin algorithm.

Case I: Low heterogeneity in both processing capacity and task workload,

Case II: High heterogeneity in processing capacity and low heterogeneity in task workload.

Case III: Low heterogeneity in node capacity and high heterogeneity in task workload.

Case IV: High heterogeneity in both node capacity and task workload.
In Case I, since the heterogeneity in node capacity is low, our proposed approach does not perform significantly better than the existing round-robin heuristic. The following graph compares the performance of round-robin heuristic and the proposed approach.

In Case II, though there is low heterogeneity in the task workload in the grid, but since the node capacities vary over a comparatively wide range, we observe significant improvement in results when we compare our approach with that of the former heuristic.

In Case III, though there is high heterogeneity in the task workload but without a high heterogeneity in the node capacity in the grid, again, the algorithm does not show much improvement over the round-robin algorithm.

In Case IV, due to high variation in individual node capacities as well as in workloads of tasks arriving in the grid, we see a marked improvement of our heuristic in comparison the round-robin algorithm as high variation in processing speeds means that we can exploit faster available machines to replicate tasks which are currently running on comparatively slower machines.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Arrival Time (millisecond)</th>
<th>Burst Time (millisecond)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job1</td>
<td>0</td>
<td>8</td>
<td>62</td>
</tr>
<tr>
<td>Job2</td>
<td>1</td>
<td>0</td>
<td>55</td>
</tr>
<tr>
<td>Job3</td>
<td>2</td>
<td>3</td>
<td>93</td>
</tr>
<tr>
<td>Job4</td>
<td>3</td>
<td>3</td>
<td>31</td>
</tr>
<tr>
<td>Job5</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job6</td>
<td>5</td>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>Job7</td>
<td>6</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>Job8</td>
<td>7</td>
<td>8</td>
<td>56</td>
</tr>
<tr>
<td>Job9</td>
<td>8</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>Job10</td>
<td>9</td>
<td>1</td>
<td>54</td>
</tr>
</tbody>
</table>

Table 4.1: Jobs Scenario Table for the Priority Based Multiple Queues.
6. Conclusions

The following paper describes a novel approach to schedule tasks efficiently in a grid environment, without having prior information on workload of incoming tasks. We propose an enhancement to the existing round-robin heuristic by prioritizing tasks eligible for replication. The approach is based on exploiting information on processing capability of individual grid resources and applying replication on tasks assigned to the slowest processors. Future work will focus on designing a scalable architecture for scheduling tasks on grids comprising of nodes of the order of thousands, helping us to assess the performance of the heuristic to a better extent. In the present approach, we have ignored the costs involved in transporting tasks to their designated machines. To take the communication costs involved into account, along with the obvious processing costs of these tasks, is intended to be taken up as a subject for our future work.

References


[8] Agarwal, A., Kumar, P.: Economical Duplication Based Task Scheduling


Abstract-On-demand routing protocols use route caches to make routing decisions. Due to mobility, cached routes easily become stale. To address the cache staleness issue, prior work in DSR used heuristics with ad hoc parameters to predict the lifetime of a link or a route. However, heuristics cannot accurately estimate timeouts because topology changes are unpredictable. In this paper, we propose proactively disseminating the broken link information to the nodes that have that link in their caches. We define a new cache structure called a cache table and present a distributed cache update algorithm. Each node maintains in its cache table the information necessary for cache updates. When a link failure is detected, the algorithm notifies all reachable nodes that have cached the link in a distributed manner. The algorithm does not use any ad hoc parameters, thus making route caches fully adaptive to topology changes. We show that the algorithm outperforms DSR with path caches and with Link-MaxLife, an adaptive timeout mechanism for link caches. We conclude that proactive cache updating is key to the adaptation of on-demand routing protocols to mobility.

Keywords: Mobile ad hoc networks, On-demand routing protocols, Mobility, Distributed cache Modernizing.

I. INTRODUCTION

In a mobile ad hoc network, nodes move arbitrarily. Mobility presents a fundamental challenge to routing protocols. Routing protocols for ad hoc networks can be classified into two major types: proactive and on-demand. Proactive protocols attempt to maintain up-to-date routing information to all nodes by periodically disseminating topology updates throughout the network. In contrast, on-demand protocols attempt to discover a route only when a route is needed. To reduce the overhead and latency of initiating a route discovery for each packet, on-demand routing protocols use route caches. Due to mobility, cached routes easily become stale. Using stale routes causes packet losses, and increases latency and overhead. In this paper, we investigate how to make on-demand routing protocols adapt quickly to topology changes. This problem is important because such protocols use route caches to make routing decisions; it is challenging because topology changes are frequent.

To address the cache staleness issue in DSR (the Dynamic Source Routing protocol) [6], [8], prior work [4], [11], [9] used adaptive timeout mechanisms. Such mechanisms use heuristics with ad hoc parameters to predict the lifetime of a link or a route. However, a predetermined choice of ad hoc parameters for certain scenarios may not work well for others, and scenarios in the real world are different from those used in simulations. Moreover, heuristics cannot accurately estimate timeouts because topology changes are unpredictable. As a result, either valid routes will be removed or stale routes will be kept in caches.

To evict stale routes faster, DSR with path caches uses a small cache size. However, as traffic load or network size increases, small caches will cause route re-discoveries, because more routes need to be stored, but small caches cannot hold all useful routes. If the cache size is set large, more stale routes will stay in caches because FIFO replacement becomes less effective. It was shown that path caches with unlimited size perform much worse than caches with limited size, due to the large amount of ROUTE ERRORS caused by the use of stale routes [4].

In this paper, we propose proactively disseminating the broken link information to the nodes that have that link in their caches. Proactive cache updating is key to making route caches adapt quickly to topology changes. It is also important to inform only the nodes that have cached a broken link to avoid unnecessary overhead. Thus, when a link failure is detected, our goal is to notify all reachable nodes that have cached the link about the link failure.

We define a new cache structure called a cache table to maintain the information necessary for cache updates. A cache table has no capacity limit; its size increases as new routes are discovered and decreases as stale routes are removed. Each node maintains in its cache table two types of information for each route. The first type of information is how well routing information is synchronized among nodes on a route: whether a link has been cached in only upstream nodes, or in both upstream and downstream nodes, and neither. The second type of information is which neighbor has learned which links through a ROUTE REPLY. Thus, for each link in a node’s cache, the node knows which neighbor nodes have cached that link. Therefore, topology propagation state, the information necessary and sufficient to remove stale routes, is kept in a distributed manner.

We design a distributed algorithm that uses the information kept by each node to achieve distributed cache updating. When a link failure is detected, the algorithm notifies selected neighborhood nodes about the broken link: the closest upstream and/or downstream nodes on each route containing the broken link, and the neighbors that learned the link through ROUTE REPLIES. When a node receives a notification, the algorithm notifies selected neighbors. Thus, the broken link information will be quickly propagated to all reachable nodes that need to be notified.

Our algorithm has the following desirable properties:
- Distributed: The algorithm uses only local information and communicates with neighborhood nodes; therefore, it is scalable with network size.
- Adaptive: The algorithm notifies only the nodes that have cached a broken link to update their caches; therefore, cache update overhead is minimized.
- Proactive on-demand: Proactive cache updating is triggered on-demand, without periodic behavior.
- Without ad hoc mechanisms: The algorithm does not use any ad hoc parameters, thus making route caches fully adaptive to topology changes.

Each node gathers the information about which node learns which link through forwarding packets, not through promiscuous mode, which is an optimization for DSR [10]. To handle situations where promiscuous mode is used, we combine our algorithm and the secondary cache used in DSR with path caches, without any modification to the algorithm.

We evaluate the algorithm with and without promiscuous mode through detailed simulations. We show that, under non-promiscuous mode, the algorithm outperforms DSR with path caches by up to 19% and DSR with Link-MaxLife [4] by up to 41% in packet delivery ratio. Under promiscuous mode, the algorithm improves packet delivery ratio by up to 7% for both caching strategies and reduces latency by up to 27% for DSR with path caches and 49% for DSR with Link-MaxLife.

Our contributions are threefold. First, we addressed the cache updating issue of on-demand routing protocols. Second, we show that proactive cache updating is more efficient than adaptive timeout mechanisms. Finally, we conclude that proactive cache updating is key to the adaptation of on-demand routing protocols to mobility.

2. RELATED WORK
Maltz et al. [10] were the first to study the cache performance of DSR. They found that the majority of ROUTE REPLIES are based on cached routes, and only 59% of ROUTE REPLIES carry correct routes. They also observed that even ROUTE REPLIES from the target are not 100% correct, since routes may break while a ROUTE ERROR is sent back to the source node. They concluded that efficient route maintenance is critical for all routing protocols with route caches.

Holland and Vaidya [3] showed that stale routes degrade TCP performance. They observed that TCP experiences repeated route failures due to the inability of a TCP sender’s routing protocol to quickly recognize and remove stale routes from its cache. This problem is complicated by allowing nodes to respond to route discovery requests with routes from their caches, because they often respond with stale routes. Perkins et al. [14] showed the impact of stale routes on DSR.

Hu and Johnson [4] studied the design choices for cache structure, cache capacity, and cache timeout. They proposed several adaptive timeout mechanisms for link caches. In Link-MaxLife [4], the timeout of a link is chosen according to a stability table in which a node records its perceived stability of each other node. A node chooses the shortest-length path that has the longest expected lifetime.

3. THE DYNAMIC SOURCE ROUTING PROTOCOL
A. Overview of DSR
DSR consists of two on-demand mechanisms: Route Discovery and Route Maintenance. When a source node wants to send packets to a destination to which it does not have a route, it initiates a Route Discovery by broadcasting a ROUTE REQUEST. The node receiving a ROUTE REQUEST checks whether it has a route to the destination in its cache. If it has, it sends a ROUTE REPLY to the source including a source route, which is the concatenation of the source route in the ROUTE REQUEST and the cached route. If the node does not have a cached route to the destination, it adds its address to the source route and rebroadcasts the ROUTE REQUEST. When the destination receives the ROUTE REQUEST, it sends a ROUTE REPLY containing the source route to the source. Each node forwarding a ROUTE REPLY stores the route starting from itself to the destination. When the source receives the ROUTE REPLY, it caches the source route.

In Route Maintenance, the node forwarding a packet is responsible for confirming that the packet has been successfully received by the next hop. If no acknowledgement is received after the maximum number of retransmissions, the forwarding node sends a ROUTE ERROR to the source, indicating the broken link. Each node forwarding the ROUTE ERROR removes from its cache the routes containing the broken link.

B. Route Caching in DSR
DSR uses path caches [1] or link caches [4]. In a path cache, a node stores each route starting from itself to
another node. In a link cache, a node adds a link to a topology graph, which represents the node’s view of the network topology. Links obtained from different routes can form new routes. Thus, link caches provide more routing information than path caches.

A node learns routes through forwarding ROUTE REPLIES and data packets, or by overhearing packets when promiscuous mode is used [10]. DSR does not cache the source route accumulated in a ROUTE REQUEST, since ROUTE REQUESTS are broadcast packets and thus links discovered may not be bi-directional [8]. Due to the same reason, when a node forwards a ROUTE REPLY, it caches only the links that have been confirmed by the MAC layer to be bi-directional [8], which are the downstream links starting from the node to a destination. When forwarding a data packet, a node caches the upstream links as a separate route. After initiating a Route Discovery, a source node may learn many routes returned either by intermediate nodes or by the destination; it will cache all those routes. Thus, DSR aggressively caches and uses routing information.

Besides Route Maintenance, DSR uses two mechanisms to remove stale routes. First, a source node piggybacks on the next ROUTE REQUEST the last broken link information, which is called a GRATUITOUS ROUTE ERROR. Although this optimization helps remove stale routes from more caches, GRATUITOUS ROUTE ERRORS are not able to reach all nodes whose caches contain the broken link, because some ROUTE REQUESTS will not be further propagated due to the use of responding to ROUTE REQUESTS with cached routes. Second, DSR uses heuristics: a small cache size with FIFO replacement for path caches and adaptive timeout mechanisms for link caches [4], where link timeouts are chosen based on observed link usages and breakages.

4. THE DISTRIBUTED CACHE MODERNIZING ALGORITHM

In this section, we first describe the cache staleness issue. We then give the definition of a cache table and present two algorithms used to maintain the information for cache updates. Finally, we describe our distributed cache update algorithm in detail.

A. Problem Statement

On-demand Route Maintenance results in delayed awareness of mobility, because a node is not notified when a cached route breaks until it uses the route to send packets. We classify a cached route into three types:

- pre-active, if a route has not been used;
- active, if a route is being used;
- post-active, if a route was used before but now is not.

It is not necessary to detect whether a route is active or post-active, but these terms help clarify the cache staleness issue. Stale pre-active and post-active routes will not be detected until they are used. Due to the use of responding to ROUTE REQUESTS with cached routes, stale routes may be quickly propagated to the caches of other nodes. Thus, pre-active and post-active routes are important sources of cache staleness.

We show an example of the cache staleness issue. In Figure 1, assume that route ABCDE is active, route FGCDH is post-active, and route IGCDJ is pre-active. Thus, node C has cached both the upstream and the downstream links for the active and post-active routes, but only the downstream links, CDJ, for the pre-active route. When forwarding a packet for source A, node C detects that link CD is broken. It removes stale routes from its cache and sends a ROUTE ERROR to node A. However, the downstream nodes, D and E, will not know about the broken link. Moreover, node C does not know that other nodes also have cached the broken link, including all the nodes on the post-active route, F, G, D, and H, and the upstream nodes on the pre-active route, I and G.

![Fig.1. An Example of Routing Caching in DSR.](image)

B. Assumption

Promiscuous mode [10] disables the network interface’s address filtering function and thus causes a protocol to receive all packets overheard by the interface. Since it is impossible to know which neighbor overhears which link, we do not maintain such information in a cache table. To handle promiscuous mode, we use a secondary cache to store overhead routes, without any modification to the cache update algorithm.

C. The Definition of a Cache Table

It was shown that no single cache size provides the best performance for all mobility scenarios [4]. Thus, we design a cache table that has no capacity limit. Without capacity limit allows DSR to store all discovered routes and thus reduces route discoveries. The cache size increases as new routes are discovered and decreases as stale routes are removed.

There are four fields in a cache table entry:

- Route: It stores the links starting from the current node to a destination or from a source to a destination.
- SourceDestination: It is the source and destination pair.
- DataPackets: It records whether the current node has forwarded 0, 1, or 2 data packets. It is 0 initially, incremented to 1 when the node forwards the first data packet, and incremented to 2 when it forwards the second data packet.
- ReplyRecord: This field may contain multiple entries and has no capacity limit. A ReplyRecord entry has two fields: the neighbor to which a ROUTE REPLY is forwarded and the route starting from the current node to a destination. A ReplyRecord entry will be removed in two cases: when the second field contains a broken
link, and when the concatenation of the two fields is a sub-route of the source route, which starts from the previous node in the source route to the destination of the data packet.

5. PERFORMANCE EVALUATION

A. Evaluation Methodology

We compared our algorithm called DSR-Update to DSR with path caches and with Link-MaxLife under both promiscuous and non-promiscuous mode. When promiscuous mode (also called tapping) was not used, we did not use GRATUITOUS ROUTE ERRORS, since we wanted to use the algorithm as the only mechanism to remove stale routes. When promiscuous mode was used, we used all optimizations for the three caching strategies.

B. Simulation Results

1) Packet Delivery Ratio: Figure 13 (a)–(c) show packet delivery ratio. Without promiscuous mode, DSR-Update outperforms DSR with path caches by up to 19% and Link-MaxLife by up to 41%. The improvement increases as mobility, traffic load, or network size increases. As mobility increases, more routes will become stale; therefore, the advantages of fast cache updating become more significant. As traffic load increases, stale routes will adversely affect more traffic sources; proactive cache updating reduces packet losses from more sources. Proactive cache updating is also important for large networks, because as network size increases, more nodes will cache stale routes.

2) Packet Delivery Latency: Figure 14 shows packet delivery latency. Without promiscuous mode, DSR-Update reduces latency by up to 54% of DSR with path caches. Since detecting link failures is the dominant factor of delivery latency, the reduction in latency further demonstrates the effectiveness of the algorithm. Moreover, the reduction increases as mobility, traffic load, or network size increases, because quick removing stale routes reduces link failure detections by multiple flows.

6. CONCLUSIONS

In this paper, we presented the first work that proactively updates route caches in an adaptive manner. We defined a new cache structure called a cache table to maintain the information necessary for cache updates. We presented a distributed cache update algorithm that uses the local information kept by each node to notify all reachable nodes that have cached a broken link. The algorithm enables DSR to adapt quickly to topology changes.

We show that, under non-promiscuous mode, the algorithm outperforms DSR with path caches by up to 19% and DSR with Link-MaxLife by up to 41% in packet delivery ratio. It reduces normalized routing overhead by up to 35% for DSR with path caches. Under promiscuous mode, the algorithm improves packet delivery ratio by up to 7% for both caching strategies, and reduces delivery latency by up to 27% for DSR with path caches and 49% for DSR with Link-MaxLife. The improvement demonstrates the benefits of the algorithm. Although the results were obtained under a certain type of mobility and traffic models, we believe that the results apply to other models, as the algorithm quickly removes stale routes no matter how nodes move and which traffic model is used.

The central challenge to routing protocols is how to efficiently handle topology changes. Proactive protocols periodically exchange topology updates among all nodes, incurring significant overhead. On-demand protocols avoid such overhead but face the problem of cache updating. We show that proactive cache updating is more efficient than adaptive timeout mechanisms. Our work combines the advantages of proactive and on-demand protocols: on-demand link failure detection and proactive cache updating. Our solution is applicable to other on-demand routing protocols. We conclude that proactive cache updating is key to the adaptation of on-demand routing protocols to mobility.

REFERENCES


Improved Congestion Control for Packet Switched Networks and the Web Servers

G. Thilip Kumar and Dr. R. Balasubramanian

Abstract --- Cluster based web server has become a critical issue in various concert. Since user-level communication is an effective technique to reduce the intra-cluster communication overhead, cluster-based Web server design utilizing user-level communication mechanisms has been a popular research focus in recent times. In this context, the overall objective of this research is to explore several design issues in order to enhance the performance of cluster-based network servers based on user-level communication mechanisms. Due to their complexity, the performance of cluster-based Web servers is dependent on various design choices and demands imposed on the systems. Therefore, we investigate several design issues with different system environments as explained below. First, we propose a coscheduled server model, which coschedules the communication processes of a request. Its main objective is to minimize the response time of requests that need intra-cluster communication. It has two familiar techniques called Dynamic Coscheduling and DCS with immediate blocking. Secondly, SSL is providing the cluster-based application servers and it proposes a SSL with BF. Hence, improving the concert of SSL-enabled application servers is serious for designing scalable and high concert data center. Finally, it proposes a NIC data caching scheme, and it improves the distributed Web server’s performance. Although NIC memory is traditionally used for communication, it can be used for improving performance as well because modern NICs are equipped with much larger amounts of memory.

To investigate these techniques, it develops a widespread recreation testbed that captures the underlying communication layer in a cluster, the coscheduling algorithms, and the characteristics of a distributed Web server model. Extensive experiments of three proposed schemes using this testbed show that performance of cluster-based Web servers can significantly increase by employing these techniques.

Keywords---Cluster Based Web, Dynamic Coscheduling, NIC, Secure Socket Layer.

I. INTRODUCTION

The front-end web server provides static or simple dynamic services. The web resources provided by the first tier are usually open to the public and, thus, don’t require authentication or data encryption. Hence, the average latency of client requests in this layer is usually shorter than in the application servers. The mid-tier, called the application server, is located between the web servers and the back-end database [1]. The application server has a separate load balancer and a security infrastructure such as a firewall and should be equipped with a support for databases, transaction management, communication, legacy data, and other functionalities [2]. After receiving a client’s request, an application server parses and converts it to a query. Then, it sends the generated query to a database and gets back the response from the database. Finally, it converts the response into an html based document and sends it back to the client.

The application server provides important functionalities for online business such as online billing, banking, and inventory management [3]. Therefore, the majority of the content here is generated dynamically and requires an adequate security mechanism [6]. The back-end database layer houses the most confidential and secure data. The main communication overhead of a database layer is the frequent disk access through the Storage Area Network [4].

II. RELATED WORK

We summarize the prior studies related to this research. In a single Web server environment, the cost of the SSL layer was studied by Apostolopoulos et al [8], using the Netscape Enterprise Server and Apache Web server, and it was shown that the session reuse is critical for improving the performance of Web servers.

Recent studies on data centers have focused on cluster-based Web servers and the following works are related to our research. M. Aron et al. have proposed the backend request forwarding scheme in cluster-based Web servers for supporting HTTP1.1 persistent connections [9]. The client requests are directed by a content-blind webswitch to a Web server in the cluster by a simple distribution scheme such as Round-Robin DNS. The first node that receives the request is called the initial node [5]. The initial node parses the request and determines whether to service it by itself or forward it to another node based on the cache and load balance information. The forwarded request is sent back to the initial node for responding to the client. However, this study does not consider the impact of user-level communication and SSL-enabled application servers.

C. Azma et al. have explored the characteristics of several Web sites, including auction, online bookstore and bulletin board sites, using synthetic benchmarks. In their study, an online bookstore benchmark reveals that CPU in the database server is the bottleneck, while auction and bulletin board sites show that CPU in the Web server is the bottleneck. E. Cecchet et al [13]. examined the performance and scalability issues in Enterprise JavaBeans (EJB) applications. They modeled an online auction site like eBay and experimented it by several EJB implementations. Their test shows that CPU on EJB application server is the performance obstacle. In addition, the network is also saturated for some services [7]. However, none of these
studies has investigated the application server performance with SSL offering.

A. Coscheduled Web Servers

Request response time significantly affects user satisfaction. A significant volume of work has already been done regarding improvements in Web server throughput. However, latency time has garnered relatively little attention although it is also an important aspect in designing cluster-based Web servers. The steady growth of dynamic Web content has made latency reduction even more critical [10]. In addition to user-level communication, coscheduling of communicating processes has been used as an effective approach in minimizing the overall execution time of parallel application on clusters. Because the PRESS Web server and other distributed server models indicate a significant amount of intra-cluster communication to service client requests and to update caching information, we propose a coscheduled distributed Web server model to facilitate faster communication. Coscheduling techniques are employed between the two threads which handle the intra-cluster communications in the sending and receiving nodes. Intra-cluster communication occurs when the requested file resides on a remote server cache in distributed Web servers. Since reading from the remote data cache takes much less time than accessing the disk, the request is forwarded to the server which has the requested file in the data cache. We prepare to examine the impact of coscheduling techniques on Web server response time.

B. Load Balancing of the Secure Servers

Application servers in a cluster-based Web server handle dynamic and sensitive Web content that requires protection from snooping, fiddling and counterfeit. Although SSL is the best protocol to offer a secure channel between a client and an application server, adopting SSL results in degrading the server performance considerably due to its high overhead. This overhead becomes even more severe when it is deployed in application servers, since both the dynamic content of a Web server and the cryptographic algorithms of SSL require high CPU computation time [11].

Considering the fact that most dynamic content requires protection from unauthorized clients, performance improvement of SSL-enabled application servers is a pressing research topic. Hence, it investigates the impact of SSL providing in cluster based application servers, and proposes a backend forwarding mechanism by exploiting the low overhead of user-level communication [12]. If we compare the Round Robin, SSL-with-session and SSL-with-bf, RR model is commonly used in Web clusters and SSL-with-session uses a more refined distribution algorithm. So we propose the model SSL with BF is best to improve the consumption through all nodes.

C. The NIC Caching Scheme

A user-level communication protocol that is employed in a cluster-based Web server requires an intelligent/programmable Network Interface Card (NIC) to facilitate direct communication between a user and a Network Interface (NI). NICs have, therefore, a profound impact on the overall communication performance. Traditionally, the on-chip/local memory of a NIC is usually small in size and is used primarily for establishing and maintaining connections in a cluster. However, recent trends show that many modern programmable NICs come with a large amount of on-chip memory compared to the Ethernet-based NICs. Thus, we propose a NIC caching scheme that takes advantage of the large NIC memory for enhancing performance. In our proposed scheme, the NIC memory is used by Web servers to cache Web content locally or remotely. That is, requested data can be read from the local NIC memory or from the remote NIC memory. A read from the local NIC memory is done when a requested file is not in the local cache, but in the local NIC memory. This is beneficial since the NIC memory can be used as extended memory. A read from a remote NIC memory occurs when a remote node serves a requested file in its NIC memory and forwards to it the requesting (initial) node. This is beneficial because it reduces the PCI traffic and DMA transfer time. We use three Web server traces (i.e., Penn State CSE (CSE), UC Berkeley (UCB) and Penn State University (PSU) traces) to analyze the impact of our proposed NIC caching schemes.

III. IMPLEMENTATION

We conduct an in-depth performance evaluation of the NIC caching scheme to examine various design issues. The performance of the exclusive and inclusive caching schemes is analyzed based on experiments using the CSE, UCB and PSU workloads. Particularly, we focus on identifying the characteristics of the workload which have significant impacts on performance. We also examine the impacts of bandwidth in intra-cluster communications, number of server nodes in the cluster, and update interval of cache replacement on the performance of a cluster-based web server.

- In our implementation, the files are uploaded into the cluster web server in an encrypted format by administrator.
- The user downloads the required encrypted file from anyone of the server based on the response time of each server.
- The time required to retrieve a file from each server is calculated.

The file is downloaded from the server which takes minimum time for retrieval.
IV. SYSTEM ARCHITECTURE

The Figure 4.1 depicts the typical architecture of a cluster-based data center or network server consisting of three layers: front-end Web server, mid-level application server, and back-end database server. A web server layer in a data center is a web system architecture that consists of multiple server nodes interconnected through a System Area Network (SAN). The web server presents the clients a single system view through a front-end web switch, which distributes the requests among the nodes. A request from a client goes through a web switch to initiate a connection between the client and the web server. When a request arrives at the web switch, the web switch distributes the request to one of the servers using either a content-aware or a content-oblivious distribution.

V. CONCLUSION

Web servers have become a serious issue in coping with the increasing use of network-based services. The critical nature of many online transactions and distributed services mandate the increasing use of network-based services. The critical nature of load distribution among the cluster nodes by exploiting the low overhead of user-level communication. In SSL, servers can use the session reuse scheme; where a client’s previous session information is used to avoid the expensive authentication process. The back-end forwarding mechanism uses a session-aware distribution policy at a web switch to maximize the possibility that each server takes advantage of session reuse. Our results indicate that the backend forwarding scheme is quite useful to enhance the performance of SSL-enable servers.

Secondly, we investigated the effect of SSL proposing in cluster-based Web servers, while a typical Web server uses only the main memory as its data cache. We examine two alternative schemes; the exclusive and inclusive caching schemes. The exclusive caching scheme uses the NIC memory as an extended cache for the main memory cache. This scheme increases the cache hit ratio, and thus reduces the number of disk accesses. The exclusive scheme is particularly effective when the number of Web data items is large and the popularity skewness is low, as in the UCB trace. The inclusive caching scheme maintains a copy of the most frequently accessed data items (or files) in the NIC memory. Thus, the NIC cache is a subset of the main memory cache. The main advantage of this scheme is to reduce the intra-cluster communication traffic and the DMA latency. The inclusive scheme is particularly effective when the size of Web content is large and the popularity skewness is high.

A. Contributions

The major contributions of this thesis are summarized as follows. First, we proposed a coscheduled Web server model, which adopts coscheduling mechanisms between communicating processes of Web servers to reduce the response time of requests. The round trip time is reduced, the communicating processes of the requests are scheduled by the operating system as soon as possible. Also, the round trip time is reduced when the coscheduling algorithms reduce the context switches of processes. We analyzed four different Web server models: PRESS over VIA, coscheduled PRESS with DCS, coscheduled PRESS with DCS and blocking, and Adaptive, in 16-node and 32-node configurations and found that response times can be reduced significantly, especially in lightly loaded systems.

Secondly, we investigated the effect of SSL proposing in cluster-based application servers, and proposed a backend forwarding mechanism to provide higher throughput and even load distribution among the cluster nodes by exploiting the low overhead of user-level communication. In SSL, servers can use the session reuse scheme; where a client’s previous session information is used to avoid the expensive authentication process. The back-end forwarding mechanism uses a session-aware distribution policy at a web switch to maximize the possibility that each server takes advantage of session reuse. Our results indicate that the backend forwarding scheme is quite useful to enhance the performance of SSL-enable servers.

Third, we proposed to use the NIC cache as an extended cache, while a typical Web server uses only the main memory as its data cache. We examine two alternative schemes; the exclusive and inclusive caching schemes. The exclusive caching scheme uses the NIC memory as an extended cache for the main memory cache. This scheme increases the cache hit ratio, and thus reduces the number of disk accesses. The exclusive scheme is particularly effective when the number of Web data items is large and the popularity skewness is low, as in the UCB trace. The inclusive caching scheme maintains a copy of the most frequently accessed data items (or files) in the NIC memory. Thus, the NIC cache is a subset of the main memory cache. The main advantage of this scheme is to reduce the intra-cluster communication traffic and the DMA latency. The inclusive scheme is particularly effective when the size of Web content is large and the popularity skewness is high.

REFERENCES


0974-9713/CIIT-II-5890/04/$20/$100 © 2015 CiIT Published by the Coimbatore Institute of Information Technology

G. Thilipkumar, Ph.D., Research Scholar (Full time), PG & Research Department of Computer Science, J.J. College of Arts & Science, Pudukkottai, Tamilnadu, India. D.O.B : 25-01-1982.

He has completed M.Sc., (2005) in Computer Science from Bharathidasan University, Tiruchirappalli, Tamilnadu, India, M.Phil., (2007) in Computer Science from Periyar University, Salem, Tamilnadu, India and M.Ed., (2012) from Tamilnadu Teacher Education University, Chennai, Tamilnadu, India. Now, he is doing Ph.D., (registered in 2013) in Computer Science as a full time researcher at J.J. College of Arts & Science, Pudukkottai from Bharathidasan University, Tiruchirappalli, Tamilnadu, India.
Distributed Computing Research Issues in Grid Computing

G. Thilipkumar and Dr.R. Balasubramanian

Abstract--- Grid computing raises challenging issues in many areas of computer science, and especially in the area of distributed computing, as Computational Grids cover increasingly large networks and span many organizations. In this paper we briefly motivate Grid computing and introduce its basic concepts. We then highlight a number of distributed computing research questions, and discuss both the relevance and the shortcomings of previous research results when applied to Grid computing. We choose to focus on issues concerning the dissemination and retrieval of information and data on Computational Grid platforms. We feel that these issues are particularly critical at this time, and as we can point to preliminary ideas, work, and results in the Grid community and the distributed computing community. This paper is of interest to distributing computing researchers because Grid computing provides new challenges that need to be addressed, as well as actual platforms for experimentation and research.

Keywords: Distributed Computing, Grid Computing, Fabric, Transport, Virtual Organization

I. INTRODUCTION

As computation, storage, and communication technologies steadily improve, increasingly large, complex, and resource-intensive applications are being developed both in research institutions and in industry. It is a common observation that computational resources are failing to meet the demand of those applications. The power of network, storage, and computing resources is projected to double every 9, 12, and 18 months, respectively. As noted in [15], those three constants have important implications. Anticipating the trends in storage capacities (and price), application developers and users are planning increasingly large runs that will operate on and generate petabytes of data. Although microprocessors are reaching impressive speeds, in the long run they are falling behind storage. As a result, it is becoming increasingly difficult to gather enough computational resources for running applications at a single location. Fortunately, improvements in wide-area networking make it possible to aggregate distributed resources in various collaborating institutions and to form what have come to be known as Computational Grids (or Grids). To date, most Grid applications have been in the area of scientific computing as scientists world-wide are resorting to numerical simulations and data analysis techniques to investigate increasingly large and complex problems. Recently, Grid computing has been identified as a critical technology by industry for enterprise computing and business-to-business computing [11].

The term Grid was coined in the late 90s [13] to describe a set of resources distributed over wide area networks that can support large-scale distributed applications. The analogy likens the Grid to the electrical power grid: access to computation and data should be as easy, pervasive, and standard as plugging in an appliance into an outlet. This analogy is appealing and was made as early as 1965 [8]. The term Grid computing has been widely adopted (e.g. see articles in the New York Times on August 9th and 12th, 2001). In fact, the term has been used in so many contexts that it has become difficult to get a clear picture of what Grid computing really is.

In the foundational paper “The Anatomy of the Grid” [3], Foster, Kesselman, and Tuecke attempt to address this problem by (re-)defining the Grid problem as coordinated resource sharing and problem solving in dynamic, multi-institutional, virtual organizations. This concept of a virtual organization (VO) is central to Grid computing. A simplified view is that a VO is a set of participants with various relationships that wish to share resources to perform some task. In that paper, Foster et al. argue that the Grid problem is thus central not only to “e-science”, but also to industry, where the coordination of distributed resources both within and across organizations is increasingly important.

Grid computing has been the focus of a tremendous amount of research and development effort, both in research institutions and in industry. Even though the technology is in its early development stages and is still evolving rapidly, Grid systems are being deployed and used worldwide. This situation creates a great opportunity for computer science researchers in several areas for two reasons. First, many crucial computer science research questions need to be answered in order to deploy and operate Grids effectively. Second, now that basic infrastructure elements are in place [6], the Grid has become a viable platform for such research. Indeed, as communities of users start running applications, large

G. Thilipkumar, Research Scholar, J.J. College of Arts and Science, Bharathidasan University, Trichy.
Dr.R. Balasubramanian, Research Guide, J.J. College of Arts and Science, Bharathidasan University, Trichy.

amounts of empirical trace data are becoming available for determining relevant characteristics of Grid platforms; examples include usage patterns, resource availability, and resource contention. These data make it possible to build increasingly realistic models that are critical for conducting Grid computing research.

II. RELATED WORK

In the previous section we have motivated the need for Grid computing. We give a brief introduction to basic concepts, to set the stage for the later sections of this paper. This material borrows heavily from the two foundational papers. The Anatomy of the Grid [11] and The Physiology of the Grid [10] which make fundamental contributions by defining the field and providing a common vocabulary.

Let us state again the Grid problem as presented in [9]: coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. This definition expresses several distinct dimensions of the Grid problem:

- The word “resource” is to be taken in a broad sense, to include data, computers, scientific instruments, software, etc.
- Sharing must be “coordinated” in that resources together with their providers/consumers are clearly defined, and in that multiple resources may need to be organized in an integrated fashion to achieve various qualities of service. Achieving this coordination involves the establishment and enforcement of sharing agreements [7].
- The ability to negotiate resource sharing agreements in a dynamic and flexible fashion enables a wide variety of problem solving methodologies, ranging from collaborative engineering to distributed data mining.
- Membership in a VO is “dynamic” as participants may join or leave at any time. A Virtual Organization (VO) is then the set of individuals and institutions defined by this sharing. Many examples of existing or envisioned VOs show that the concept spans a broad spectrum of purpose, size, structure, and duration.

A. Grid Architecture Layers

The Grid fabric provides the lowest level of access to actual resources (e.g. computer, disk, file system, cluster of computers) and implements the mechanisms that allow those resources to be utilized. More specifically, those mechanisms must at least include state enquiry and resource management mechanisms, each of which must be implemented for a large number of native systems. The Grid connectivity layer defines communication, security, and authentication protocols required for network transactions between resources. The Grid resource layer builds on the connectivity layer to implement protocols that enable the use and sharing of individual resources. More specifically, two fundamental components are (i) information protocols for querying the state of a resource; and (ii) management protocols to negotiate access to a resource. The Grid collective layer focuses on the coordination of multiple resources. Example of functionalities include resource discovery, co-allocation, scheduling. This is the layer which is of greatest interest for the purpose of this paper as it is where many challenging distributed computing questions must be answered. Finally, the application layer is where VO applications are implemented and may use several of the previous layers. The resulting architecture is depicted in Figure 1.

![Figure 1. The layered Grid architecture and its relationship to the Internet protocol architecture reproduced from [13].](image-url)

B. Current Developments and Limitations

In this section, we describe components of the Grid infrastructure that address the management of application data and resource information, as they are the focus of our discussion in the rest of this paper.

The infrastructure that focuses on management of distributed application data is commonly labeled a Data Grid [15]. An increasing number of scientific disciplines manage large data collections generated by measurements and derivation of measurement data. As a result, many Data Grids are currently being deployed [5, 4, 15, 6, 13]. Infrastructure targeting resource information is often referred to as a Grid Information Service [7]. A number of research groups have designed and prototyped components for collecting, indexing, and publishing Grid information. The problems of indexing, discovering, and accessing such “Grid information services” is in some respects quite similar to those encountered when indexing, discovery, and accessing other data sources. However, we will see in the rest of this paper that both infrastructures...
raise a number of distinct research questions from a distributed computing perspective.

III. VIRTUAL ORGANIZATIONS

An important factor that has driven the evolution from HPC systems and applications to Grid computing is the widespread deployment of high-speed wide-area networks. The dramatic increase in network connectivity makes it feasible to consider deploying applications that tightly couple geographically distributed resources, data, and users.

This distribution has major implications when designing a software infrastructure to support problem solving activities. In a tightly coupled systems, such as a Massively Parallel Processor (MPP), it is possible to obtain an accurate picture of the global state of the system and control its components in a centralized fashion. In a VO, distributed ownership and high-latency networks render the HPC approach to system design infeasible. In order to illustrate the challenges the Grid community is facing, we present a hypothetical VO and the activities it supports.

Consider a community of thousands of users that span hundreds of research institutions worldwide and who all focus on overlapping portions of a common scientific problem. Those users and institutions form a VO. Even though the software infrastructure to support a VO of that magnitude is not fully deployed at the time of writing, a number of multi-institution projects are underway and are making rapid progress in that direction. A notable such project is GriPhyn [4], and our discussion is inspired by GriPhyn accomplishments and current developments.

The Grid available to members of our example VO consist of several types of resources: (i) a number of scientific instruments that generate raw experimental measurement data (e.g. particle collider, radio telescope); (ii) compute resources ranging from desktop workstations to clusters and MPPs, that are used to perform derivations and analysis of the measurement data as well as simulations; and (iii) storage resources on which measurement and derived data can be stored. All resources are interconnected via wide-area networks and are prone to downtime, either due to failures or to maintenance tasks.

Members of the VO, or software agents acting on their behalf, wish to perform a variety of tasks. For example:

- Publish new measurement data,
- Locate data items matching some criteria,
- Retrieve particular data items efficiently,
- Locate appropriate compute resources to run a simulation or data analysis task,
- Be constantly informed when new “relevant” data is produced,
- Publish new derived data of potential interest to other members of the VO,
- Be constantly informed of the load on a selected number of resources during the next 2 hours.

IV. GRID INFORMATION DISSEMINATION

In this section we discuss issues of scalable delivery of dynamic information about the state of a Virtual Organization.

a) Publisher/Subscriber Systems
b) Publisher/Subscriber and the Grid
c) Selectivity and Regionalism
d) Dynamic Subscriptions
e) QoS for Event Delivery

V. RETRIEVING DATA AND INFORMATION

In the previous section we have discussed systems that deliver events to subscribers. This is a flexible way to allow components to interact in large-scale, wide-area environments such as the ones that will be spanned by VOs. However, this does not imply that all interoperation in the Grid can (or should) be done via such systems. In fact, there is a clear need for enabling queries. First, some Grid resource information is static. Second, users or user agents may want to perform queries to identify all (or most) resources that fit some criteria. For instance, one may want to find all compute resources on which some specific software is currently installed and that provide at least 1GB of RAM. Also, a user needs to issue queries to discover relevant application data that is available in Grid storage devices. As we have seen in Section 4.2, events can be generated for periodic data creations. However, in a realistic VO, we also expect users and user agents to generate queries to the Data Grid to discover and retrieve archived data. The goals for Grid computing are no different from other areas: to make discovery and retrieval efficient and scalable.

VI. CONCLUSION

Grid computing is broad in its domain of application and raises research questions that span many areas of distributed computing, and of computer science in general. In this paper we have opted for providing detailed descriptions of a few issues that we feel are particularly interesting, can largely benefit from cross-fertilization with the distributed computing research community, and have already been the object of preliminary work in Grid computing. Namely, we discussed issues pertaining to the discovery and dissemination of information and data, both static and dynamic, within Virtual Organizations. We focused on two different models: subscription-based and query-based. Other authors would likely have chosen to
discuss other issues. We provide here a brief and non-

exhaustive glimpse of what those issues might be.

A fundamental concern for Grid computing is security. The current Grid Security Infrastructure (GSI) [11] supports single sign-on, delegation, and user-based trust relationships, each of which raises a number of challenging questions. Issues relating to policy have only being touched upon briefly to date. Another issue is that of scheduling for distributed applications. Application scheduling has been an active field of research for decades and the Grid poses a number of new challenges that have been identified and partially addressed [9]. A critical issue is to schedule applications that combine intensive computation with the use of data stored (and replicated) in emerging Data Grids. For instance, one question is to determine in which scenarios coupled [15] and decoupled [2] scheduling approaches are appropriate. Another issues is to ensure that scheduling agents acting on behalf of their users cooperate in order to avoid “herd behavior”. Although this problem has been identified in the context of the Grid early on, it has not been addressed. This ties into the notion of a Grid economy. Several authors argue that policies for Grid usage should be derived from economical models that are based on a commodity market. Early work in the Grid community has already evaluated a few hypotheses for defining viable Grid economy models [14].

The question of co-scheduling is that of ensuring that an application can reserve and utilize several resources simultaneously at a given time. This is a difficult problem which is important for many Grid users and applications. More generally, ways for VO participants to achieve agreement are needed. Indeed, fault-tolerance becomes a critical issue with increases in scale: large VOs mean that the probability of failure of individual components becomes significant. Agreement protocols have not been adopted by the Grid community so far. They have not yet been important for application writers, because few applications require much in the way of fault-tolerance; simply restarting a failed computation is sufficient. Also, even with an increase in VO scale, there are more scalable methods of fault-tolerance (such as rollback recovery) that would make more engineering sense to use than agreement for most applications. However, the services comprising the Grid software infrastructure itself are long-running and are key for sustaining VO activities. Although inexpensive approaches like eventual update combined with randomized scheduling have proven sufficient so far (e.g. see [3]), we believe that a number of Grid services are likely candidates for agreement protocols.

The Grid is currently being built as a concerted effort among many institutions and is already supporting leading scientific applications. In this paper we have identified several differences between the Grid and other distributed computing models and systems. We argued that those differences motivate new research questions. As more and more VOs are deployed, it will be possible to gather very large amounts of trace data concerning the social and technical interactions among VO participants. Mining that data will undoubtedly reveal crucial features of the nature of scientific collaborations that can be exploited to design appropriate distributed protocols and algorithms. Furthermore, it will be possible to construct increasingly realistic models that can be used for the evaluation of those protocols and algorithms. Finally, the Grid will provide several concrete platforms for the validation of research results in real-world scenarios. We hope that this paper provides evidence that Grid computing is an exciting area, and that it provides many opportunities for researchers in distributed computing and distributed systems to tackle new problems and to evaluate their solutions.

REFERENCES


