A Model To Improve The Performance Of Condor Through Cache Memory In Grid Environment

M. Humera Khanam
Associate Professor, Department of Computer Science and Engineering, Sri Venkateswara University, Tirupati.

N. Purandhar
PG Student, Department of Computer Science and Engineering, Sri Venkateswara University, Tirupati.

Abstract: Grid application consists of one or more tasks. These tasks are generally divided into two types such as Independent task and Workflow tasks. A number of resource brokers that act as middleware such as Condor, Nimrod-G etc are used in grid based application. Since, workflow tasks are not easily executed with these architecture, we slightly modified the Condor architecture to suit workflow tasks. This paper focuses on cache management system in Condor resource broker. By adding cache to condor architecture, it allows to store intermediate results. As a result the overhead of read and writes to central database is reduced. Therefore it improves the performance and throughput of the system.

Keywords: Cache Memory, Condor, Grid Computing, Memory Access Time, Nimrod-G, Workflow.

I. INTRODUCTION

Grid computing [1] is the current hot topic in Information Technology field. Lot of research is going for efficient utilization of grid. Most of the applications require high processing speed, lot of CPU time and memory.

Scheduling is the central part of grid computing [1]. It can be performed as simply as taking the next available resource, but often this task involves prioritizing job queues, managing the load, finding the workarounds when encountering reserved resources and monitoring progress. Job schedulers are able to submit, control and monitor the workload of jobs submitted in a network of computers.

Grid broker mediates access to distributed resources by (a) discovering suitable data sources for a given analysis scenario, (b) suitable computational resources, (c) optimally mapping analysis jobs to resources, (d) deploying and monitoring job execution on selected resources, and (e) accessing data from local or remote data source during job execution. A resource broker in a data grid must have the capability to locate and retrieve the required data from multiple data sources and to redirect the output to storage where it can be retrieved by processes downstream. It must also have the ability to select the best data repositories from multiple sites based on availability of files and quality of data transfer.

Most of the applications that use a grid have Workflow tasks. To utilize the grid for workflow tasks we have many schedulers and resource brokers like Condor and Nimrod-G. But those are not efficient for workflow tasks for instance DAG-Condor scheduler frequently queries the database for the intermediate results and the number of queries and the size of database results in extra overhead. The performance of system depends on other factors such as the numbers of machines in the cluster and the workload. This creates problems like congestion, and degradation of throughput and performance. To overcome the above problem we proposed architecture by adding cache to store intermediate results and these results also stored in central database which contains old results. To increase the efficiency and performance we have proposed cache management system in DAG-Condor resource broker.
II. RELATED WORK

A Workflow is a collection of Steps and data that define the paths that can be taken to complete a task. Workflows may contain activities such as displaying content to users, collecting information from users or computer systems, performing calculations, and sending messages to external computer systems. To maintain the workflow jobs we have collection of resource brokers such as Nimrod-G, Condor-G, AppLeS, Gridbus Broker, Globus.

The Nimrod-G [2] is a Grid resource broker that allows managing and steering task farming applications on computational Grids. It uses an economic model for resource management and scheduling. Nimrod-G provides resource discovery, resource trading, scheduling, resource staging on Grid nodes, result gathering, and final presentation to the user. Nimrod-G uses GRid Architecture for Computational Economy (GRACE) services to dynamically trade with resource owner agents to select appropriate resources. It follows the hierarchical and computational market model in resource management. It uses the services of Grid middleware systems such as Globus and Legion for resource discovery and uses either a network directory or object model based data organization. Nimrod/G is a resource management system with a focus on computational economy and schedules tasks based on their deadlines and budgets. Nimrod/G also addresses issues of scheduling single jobs, and does not address the requirements of workflow applications.

The Condor is a Meta computing system that uses computer idle time to run jobs in a network. Given the computing resources of an institution, Condor seeks to maximize job throughput, without disturbing human interaction. The Condor environment follows a layered architecture and offers powerful and flexible resource management services for sequential and parallel applications. The Condor has been extended to support submission of jobs to resources Grid-enabled using Globus services. The matchmaker is responsible for initiating contact between compatible agents.

The Condor-G [6] is the job management part of Condor. Condor-G helps us to submit jobs into a queue, have a log detailing the life cycle of our jobs, manage all input and output files, along with everything else you expect from a job queuing system. Condor-G gets its name from how it talks to the resource management part. Condor-G uses the Globus Toolkit(tm) to start the job on the remote machine. Condor-G provides a "window to the Grid" for users to both access resources and manage jobs running on remote resources. Condor-G is used to look across the Grid and see instantly how the jobs are doing.

The database is resident at the server and transactions are initiated from client sites, with the server providing facilities for shared data access. Dynamic local Caching of query results at client sites can enhance the overall performance of such a system, especially when the operational data spaces of clients are mostly disjoint. In effect, such caching of locally pertinent and frequently used data constitutes a form of dynamic data replication, whereby each client dynamically defines its own data space of interest. This concept is utilized in this proposed system.

III. WORKFLOW TASK

A workflow (fig.1) is composed of connected multiple scientific tasks according to their dependencies. Workflow [11] structure indicates the temporal relationship between the tasks. In general, a workflow can be represented as a Directed Acyclic Graph (DAG)[9,10] or a non-DAG.

An acyclic digraph is a directed graph containing no directed cycles, also known as a directed acyclic graph or a "DAG." A workflow language is a particular XML notation representing the inter-task dependencies.
In DAG-based workflow, workflow structure can be categorized into three types such as sequence, parallelism, and choice. Sequence is defined as an ordered series of tasks, with one task starting after a previous task has completed. Parallelism represents tasks which are performed concurrently, rather than serially. In choice structured workflows, a task is selected to execute at run-time when its associated conditions are true. In addition to all structures contained in a DAG-based, a non-DAG workflow also includes iteration structure, in which sections of workflow tasks in an iteration block are allowed to be repeated.

3.1 Nature of scientific workflows: In workflow, DAG (Directed Acyclic Graph) [11] refers to a set of programs with dependencies between them i.e the input of some programs may depend on the output of others. This places partial constraints on the order of execution of programs. Consider the “diamond” DAG shown in Figure 2. Since programs B and C depend on the output of program A, they can be run only after A completes, but B and C themselves can be run in any order or even in parallel.

A typical scientific workflow frequently consists of a number of such DAGs with identical structures. These DAGs may share some common (global) input data. Scientific workflows are characterized by the different types of I/O performed by jobs in their lifetime. Pipeline I/O refers to the data flow that occurs between parent and child programs within a particular DAG. The term batch I/O refers to input files that are common across all DAGs in a workflow. In Figure 2 files ‘File1’ and ‘File2’ are pipeline files. ‘FileInput’ is a batch input to the DAGs in the example workflow shown in Figure 3. Batch input can occur at any level in a DAG.

In Condor, users create submit files for jobs to be executed. The submit file specifies details about the job such as the names of the executable, input and output files, and environment variables that need to be set at the time of job execution. In addition to that a user may also specify requirements and rank attributes. The requirements attribute places constraints on the machines that can run the job. Rank is
used to specify an order of preference for machines that meet the job’s requirements. A job is submitted by specifying a submit file to the `condor_submit` tool. The submit file is converted into a “classad”, which is a list of attribute-value pairs. Machines also advertise their resources using classads and can place constraints on the jobs they wish to run. In the matchmaking process each user is allotted some number of the machines in the pool based on a fair-share scheme [8]. Jobs that are selected for scheduling are first sorted in order of user priority. For each job, the list of available machines is scanned and the machine with the highest rank that satisfies the job’s requirements is chosen for execution. (This machine is called the execute machine). The matchmaking process is done periodically (typically, every five minutes).

The `condor_submit` tool only accepts jobs consisting of a single program. Workflows consisting of DAGs are submitted using a different tool called `condor_submit_dag`. This tool allows users to specify parent-child dependencies between jobs. When a workflow is submitted, the Condor DAGMan daemon is spawned, and the top-level jobs in the workflow (jobs marked ‘A’ in Figure 3) are submitted using `condor_submit`. DAGMan continuously monitors the logs produced by Condor on the submit machine (the machine to which the user submits jobs) and uses `condo_submit` to submit a child job once its parents have completed execution. So the Condor matchmaker itself has no notion of workflows and cannot consider a job for scheduling until it is ready for execution.

Before a job begins, it’s input and executable files are transferred from the submit machine to the execute machine. After a job has completed, the output files associated with that job are transferred back to the submit machine and deleted from the execute machine. When a child job is scheduled, files are again transferred from the submit machine to an execute machine (which could possibly be the same one that executed the parent job). This job scheduling and file transfer system is inefficient since the data dependencies within and across DAGs in a workflow are ignored. The main disadvantages are listed below;

- All jobs has to be submitted through submit machine only
- All the client machines send and receive intermediate results through central Database
- Congestion is high in `condor`
- Performance degrades as the number of tasks increases
- Data or internal results are not used in future
- It reduces throughput and increase the execution time.

V. PROPOSED CACHE MANAGEMENT SYSTEM IN DAG-CONDOR

The Condor architecture has a database which is connected to submit machine, so all the jobs will be submitted through the submit machine only. To overcome this problem we propose the new DAG-Condor Architecture (fig.5) in which the central database is connected to both submit and execute machine. Here we can submit any job from any part of the grid either through submit machine or execute machine.

Previously in the DAG-Condor architecture, due to the presence of a central database the overhead of accessing the database increase as and when intermediate results are generated. This in turns reduces the performance of the system. Due to the increase of read and write operations it also increases the congestion, cost of operation and increases the loss of data. To reduce the loss of data retransmission of data will increase. To reduce the overhead of read and write operations we include the high storage cache which is connected to all the execute machine clusters in the proposed system. In the cluster, we use hierarchy cache management which stores all the internal results of machines connected in cluster. These results are stored in cache as well as in central Database.
If any cluster requires the results of workflow jobs which are executed in another cluster, it will be accessed either from cache or central database based on the distance between the cache and central database. The distance between the systems will be calculated using the Link-State algorithm [16] which is explained below.

Suppose we have four clusters A, B, C, D and central database CD in grid system. Figure 7 shows that the system is connected between the clusters as follows (A,B), (B,C), (B,D), (C,D), (A,CD),(D,CD) with distance 3,1,5,1,2,1 respectively. Let job x be executed at cluster D, the result is stored in cache and central database. An execute machine in cluster B requires the result of job x which is stored in the central database and cache at cluster D. The question is that, from which memory it has to access the results. By using the Link–state Algorithm it finds that, access from cache memory at cluster D is between B, D is shorter than B to CD.
V. SYSTEM MODEL

The execution time of an application in computer mainly depends on two factors: CPU execution time and Memory access time. It is well known fact that, the cache memory helps in faster access of data, so that the access time of memory is reduced considerably. In Condor architecture also, the use of cache memory reduces the total execution time of the application by reducing the average memory access time. The mathematical model to compute Execution is described as shown below.

Calculating Execution Time:

\[
\text{Execution Time} = [\text{CPU Clock Cycles} + \text{Memory Clock Cycles}] \times \text{Clock Cycle Time}.
\]

The Memory Clock Cycles is calculated from Memory Access Time, and it is given by

\[
\text{Memory Access Time} = \text{No. of Memory Access by the Instruction} \times \text{Average Memory Access Time}
\]

Average Memory Access Time = [Hit rate \times Hit time] + [Miss rate \times Miss penalty]

VII. RESULTS

We have simulated the architecture in java which gives good results. It gives good performance when compared to the previous architecture of Condor. Here the execution time is reduced when compare to previous one, by reducing the read operation from the central database. It gives High performance and reduces the congestion due to cache management. The performance was calculated based on execution time. It reduces to a low value when compared with other approaches, thus increases the performance of the system.

Initially, I started with a 10 x 10 order matrix and 2 processors to perform distributed computing in Condor system. Then, the time taken to perform multiplication process by Condor system without cache is 743843 and Condor system with cache is 723882. The Condor system with and without cache is run on different matrices of different sizes and on varying number of processes. From all of the results, the Condor system with cache memory provides better efficiency than the other.

From table 6.2 we may give our attention that how the complexity varies with the increase of clients on one side and another with the increasing of matrices order. So it should not be avoided, rather be noted for low ordered data distributed among more clients may switched to increase the time than should be for communication complexity. However, we would like to be frankly said that the more the
machine increases the less will be the computational time which can be clearly verified by large ordered data or matrix.

Table 6.2 Result of matrix multiplication with varied matrix sizes and number of clients

<table>
<thead>
<tr>
<th>Order of Matrix(n x n)</th>
<th>Sequential Approach</th>
<th>Distributed with 2 clients</th>
<th>Distributed with 3 clients</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Without Cache</td>
<td>With Cache</td>
</tr>
<tr>
<td>20 x 20</td>
<td>5730 ms</td>
<td>2390 ms</td>
<td>1178 ms</td>
</tr>
<tr>
<td>15 x 15</td>
<td>2382 ms</td>
<td>1536 ms</td>
<td>986 ms</td>
</tr>
<tr>
<td>10 x 10</td>
<td>1345 ms</td>
<td>1284 ms</td>
<td>809 ms</td>
</tr>
<tr>
<td>05 x 05</td>
<td>597 ms</td>
<td>188 ms</td>
<td>106 ms</td>
</tr>
<tr>
<td>03 x 03</td>
<td>248 ms</td>
<td>94 ms</td>
<td>52 ms</td>
</tr>
</tbody>
</table>

VIII. CONCLUSION AND FUTURE WORK

In this paper we introduce the cache management system in DAG-Condor architecture. This makes the easy transmission of data through out the system. A major advantage is associative access to the contents of a cache, allowing effective reuse of cached information. Increased autonomy at client sites, less network traffic, and better scalability are a few other expected benefits.

Apart from the planned performance studies, many other important issues remain unexplored in this paper. We currently work on the architecture of cache management in DAG-Condor. The future implementation questions in cache are to derive suitable predicate-indexing techniques, optimization strategies, performance tuning, local index creation, and effective management of space by a client.

REFERENCES


