GHS 1.0

User Manual

Author: Ming Wu
Date: 08/18/2005

Contents

1 Installation and Running

1.1 Resource Requirement

- Software requirement

The current version of GHS 1.0 supports SunOS. The running of GHS measurement engine requires the system support the *lastcomm* Unix utility.

- Hardware requirement

GHS 1.0 does not have any specific hardware requirement.

1.2 Installation

- Down-load the source code from (TBA)

- Uncompress the source code

  $gzip –d ghs1.tar.gz
  $tar –xvf ghs1.tar.gz
  $cd ghs1

- Compile the GHS 1.0.
  $make

- Install the GHS 1.0
  $make install

1.3 Configuration

The GHS 1.0 requires the environment variable GHS_HOME to be set to the installing directory. For example, you have installed the GHS middleware to /yourdirectory/ghs1. You need to add the following line

export GHS_HOME = /your_directory/ghs1

to .profile file under your home directory if you are using ksh or sh. For csh, you need to add the following line

setenv GHS_HOME /your_directory/ghs1

to .cshrc file under your home directory.

After installation, the executables of the GHS 1.0 is located at the sub-directory bin of your installing path. It should be added into the environment variable PATH. You need to add the following line

export PATH=$PATH:$GHS_HOME/bin

to .profile file under your home directory if you are using ksh or sh. For csh, you need to add the following line

setenv PATH ${PATH}:${GHS_HOME}/bin

to .cshrc file under your home directory.

2 Introduction of GHS

Conventional performance evaluation mechanisms focus on dedicated distributed systems. Grid computing infrastructure, on another hand, is a shared collaborative environment constructed on virtual organizations. Each organization has its own resource management policy. The non-dedicated characteristic of Grid computing prevents the leverage of conventional performance evaluation systems. In this study, we introduce the Grid Harvest Service (GHS) performance evaluation and task scheduling system for solving large-scale applications in a shared environment. GHS is based on a novel performance prediction model and a set of task scheduling algorithms. GHS supports three classes of task scheduling, single task, parallel processing and meta-task.

Grid Computing introduces a great challenge in task scheduling: how to partition and schedule tasks in a large, available but shared, heterogeneous computing system. The conventional parallel processing scheduling methods cannot apply to a Grid environment where computing resources are shared and the Grid scheduler has no control over local virtual organization’s local jobs. The key to Grid task scheduling, therefore, is to estimate the availability of computing resources and to find its influence on the application performance. Some latest Grid tools have been developed to meet the need. However, these tools are for short-term resource

availability. For instance, the well-known Network Weather Service (NWS) system only predicts the availability of a computing or communication resource for the next five minutes, and its satisfactory prediction range in general is much less than the five-minute upper bound.

In this project, we present a prototype implementation of a long-term, application-level performance prediction and task scheduling system, namely Grid Harvest Service (GHS) system, for Grid computing. Here long-term means the application requiring hours or more, in contrast to minutes, sequential execution time; and application-level performance means the application turn-around time, in contrast to resource availability.

The Grid Harvest Service system comprises of five primary subsystems: performance evaluation, performance measurement, task allocation, task scheduling, and execution management. Coordinately, they provide appropriate services to harvest Grid computing. GHS performance evaluation is based on a new performance model, which is derived from a combination of stochastic analysis and direct-numerical simulation. Unlike other stochastic models, this model individually identifies the effects of machine utilization, computing power, local job service, and task allocation on the completion time of a parallel application. It is theoretically sound and practically feasible.

Utilizing the performance prediction, various partition and scheduling algorithms are developed and adopted in the partition and scheduling component, for single sequential task, single parallel task with a given number of sub-tasks, optimal parallel processing, as well as meta-task composed of a group of independent tasks. A heuristic task scheduling algorithm is proposed to find an acceptable solution with a reasonable cost. GHS uses an adaptive measurement methodology to monitor resource usage pattern, where the measurement frequency is dynamically updated according to the previous measurement history. This method reduces monitoring overhead considerably.

3 GHS 1.0 commands

GHS 1.0 includes five Unix commands to support the resource measurement (***smeas***), the system-level prediction (***syspred***), the application-level prediction (***apppred***), the meta-task scheduling (***mtsc***), and the parallel program scheduling (***pgsc***). The usage and format of these commands are given as follows.

3.1 *smeas:* resource measurement

Usage: *smeas* is a program which measures the resource availability in terms of resource utilization, job arrival rate, job service time, job service time standard deviation. The resource utilization means how much percent of resource is occupied by local jobs (We name the application to be scheduled as a remote task, in contrast to local jobs which are submitted by local users). The job arrival rate means how many jobs arrive per second during a time of period, the job service time means what is the average job length during a time of period, and the job service time standard deviation means what is the standard deviation of job length during a time of period. The collected resource information is stored in a file which default name is ghsmeasure.log. Each line records the time stamp (in hour unit) and the values of the above resource parameters during the past hour. You can set your own log file name in the ghs_config.dat.

Format: smeas

Example:
bash-2.05$ cat ghsmeasure.log | more
date meanUtil meanLambda meanServ meanStd
312111 0.252000 0.025000 14.152857 7.622611
312112 0.441240 0.013566 43.611036 3.203194
312113 0.370390 0.018182 10.270175 1.556747
312114 0.531585 0.017683 9.243261 10.244799
312115 0.410000 0.013194 8.175294 10.005561
…

3.2 *syspred:* system-level predictor

Usage: *syspred* is a program which estimates the resource availability in terms of resource utilization, job arrival rate, job service time, job service time standard deviation in a given period.

Format: *syspred* utilization|arrival|service|std period filename

The first parameter of *syspred* (utilization|arrival|service|std) gives which resource availability you want to estimate. The second parameter gives what time period in the next you want to estimate a given resource availability. The third parameter gives the log file name where the *syspred* read the performance data.

Example:
bash-2.05$ syspred arrival 10

0.079083
The above example shows the estimated job arrival rate for the next 10 hours is 0.09083. If the log file name is not given, the *syspred* will read the resource measurement history from ghsmeasure.log.

3.3 *apppred:* application-level predictor

Usage: *apppred* is a program which estimate the application execution time in terms of expected value and its standard deviation. It also gives the CDF of application execution time.

Format: apppred mapinfo.txt result.txt

The first parameter of *apppred* (mapinfo.txt) gives the name of a file where the map information is stored. The mapinfo.txt file describes how the workload of a remote task is distributed into resources and the resource information. The following is an example of a mapfile.

```
bash-2.05$ cat mapinfo.txt
hpc-01 0.225 0.015 83.49 1 500
hpc-02 0.225 0.015 83.49 1 500
hpc-03 0.225 0.015 83.49 1 500
hpc-04 0.225 0.015 83.49 3 1000
hpc-05 0.225 0.015 83.49 1 500
hpc-06 0.225 0.015 83.49 1 500
hpc-07 0.225 0.015 83.49 1 500
hpc-08 0.225 0.015 83.49 3 1000
hpc-09 0.225 0.015 83.49 1 500
hpc-10 0.225 0.015 83.49 1 500
hpc-11 0.225 0.015 83.49 1 500
hpc-12 0.225 0.015 83.49 3 1000
hpc-13 0.225 0.015 83.49 1 500
hpc-14 0.225 0.015 83.49 1 500
hpc-15 0.225 0.015 83.49 1 500
hpc-16 0.225 0.015 83.49 3 1000
```

In this mapfile, the workload of a remote task is distributed into 16 nodes, from hpc-01 to hpc-16. Each line denotes one resource. It gives the resource name, utilization, job arrival rate, job service time standard deviation, computing capacity (relative speed), and the workload of subtasks allocated to this resource.

The second parameter gives the name of a file where the prediction result is stored.

Example:

```
bash-2.05$ apppred mapinfo.txt predictionresult.txt
bash-2.05$ cat predictionresult.txt
1445.359092 686.959758
750.000000 0.090448
1000.000000 0.286710
1250.000000 0.481067
1500.000000 0.635845
1750.000000 0.748737
2000.000000 0.827856
2250.000000 0.882294
2500.000000 0.919458
2750.000000 0.944772
3000.000000 0.962026
3250.000000 0.973811
3500.000000 0.981884
3750.000000 0.987432
4000.000000 0.991256
4250.000000 0.993902
```

The above example shows the estimate result of application execution time for a given map information. In the predresult.txt file, the first line gives the expected application execution time and its standard deviation. From the second line to the last line, it gives the CDF of the application execution time. For example, the seventh line is “2000.000000, 0.827856”. That means the possibility of application execution less than 2000 is 0.827856.

3.4 *mtsc:* meta-task scheduler

Usage: *mtsc* is a program which generates a scheduling plan for a given resource set and a given meta-task.

Format: *mtsc* resopara.txt taskpara.txt result.txt schetag(0/1)

The first parameter of *mtsc* (resopara.txt) gives the name of a file where the resource information (resource name, availability, computing capacity) is described. An example of resopara.txt is given as follows.

```
bash-2.05$ cat resopara.txt
hpc-01 0.225 0.015 83.49 1
hpc-02 0.225 0.015 83.49 1
hpc-03 0.225 0.015 83.49 1
hpc-04 0.225 0.015 83.49 1
hpc-74 0.325 0.01 103.49 2
```

Each line describes one resource. It gives the resource name, utilization, job arrival rate, job service time standard deviation, and its computing capacity (relative speed). The second parameter of *mtsc* (taskpara.txt) gives the name of a file where a meta-task is described. Here is an example of taskpara.txt.

```
bash-2.05$ cat taskpara.txt
task-1 10000
task-2 10000
task-3 10000
task-4 10000
task-5 10000
task-6 10000
task-7 10000
task-8 10000
task-9 10000
task-10 10000
task-11 10000
```

The given meta-task is composed of 11 subtask. Each line gives the subtask name and its workload in terms of second.

The third parameter gives the file name where the scheduling plan is stored. The fourth parameters indicated whether the optimal scheduling algorithm or the heuristic algorithm would be used in task scheduling. 0 stands optimal scheduling and 1 stands heuristic scheduling.

Example:

```
bash-2.05$ mtsc resopara.txt taskpara.txt scheduleresult.txt 1
bash-2.05$ cat scheduleresult.txt
hpc-01: 20000.000000
  task-2
  task-8
hpc-02: 20000.000000
  task-3
  task-9
hpc-03: 20000.000000
  task-4
  task-10
hpc-04: 20000.000000
  task-5
  task-11
hpc-74: 30000.000000
  task-1
  task-6
  task-7
bash-2.05$
```

The above example shows the scheduling plan generated by mtsc (heuristic scheduling algorithm is used). In the scheduleresult.txt file, a list of resources and the subtasks allocated to these resources are given. From the schedule plan, we can see hpc-74 is allocated with three subtasks of the given meta-task, task-1, task-6, and task-7. The sum of the three subtasks is 30000 second.

3.5 *pgsc:* parallel program scheduler

Usage: *pgsc* is a program which generates a scheduling plan for a given resource set and a given meta-task.

Format: *pgsc* resopara.txt workload result.txt schetag(0/1)
The first parameter of *pgsc* (resopara.txt) gives the name of a file where the resource information (resource name, availability, computing capacity) is described. An example of resopara.txt is given as follows.

```
bash-2.05$ cat resopara.txt
hpc-01 0.225 0.015 83.49 1
hpc-02 0.225 0.015 83.49 1
hpc-03 0.225 0.015 83.49 1
hpc-04 0.225 0.015 83.49 1
hpc-74 0.325 0.01 103.49 2
```

Each line describes one resource. It gives the resource name, utilization, job arrival rate, job service time standard deviation, and its computing capacity (relative speed).

The second parameter of *pgsc* gives the total workload of this parallel program.

The third parameter gives the file name where the scheduling plan is stored. The fourth parameters indicated whether the optimal scheduling algorithm or the heuristic algorithm would be used in task scheduling. 0 stands optimal scheduling and 1 stands heuristic scheduling.

Example:

```
bash-2.05$ pgsc resopara.txt 10000 scheduleresult.txt 1
bash-2.05$ cat scheduleresult.txt
hpc-01: 1741.573028
hpc-02: 1741.573028
hpc-03: 1741.573028
hpc-04: 1741.573028
hpc-74: 3033.707886
bash-2.05$
```

The above example shows the scheduling plan generated by *pgsc* (heuristic scheduling algorithm is used). In the scheduleresult.txt file, a list of resources and the subworkloads allocated to these resources are given. From the schedule plan, we can see hpc-74 is allocated with a subworkload of 3033.707886. The other three machines are assigned with a subworkload of 1741.573028.

4 Acknowledgments

This research was supported in part by national science foundation under NSF grant CNS-0406328, ANI-0123930, and EIA-0224377.