READSD and STQA Combined Assignment

Problem Scenario

Your university is in the process of acquiring a new supercomputer to support research activities. The IT department needs to evaluate policies for the batch job submission system, with job control and accounting. To do this, it needs a simulation tool to model the behaviour of the computing platform and to explore alternative queueing/accounting strategies.

The assignment is to design, implement and test a simulator of the new computing system.

User Requirements

The simulated computing system shall be heterogeneous, containing:

- A set of "traditional" computing nodes, i.e. shared-memory multicore CPUs;
- A set of "accelerated" compute nodes, same as above but with 2 attached GPUs dedicated to computations;

There shall be a total of at least 128 nodes with at least 16 processor cores per node. Among the available nodes, at least 8 shall be equipped with GPUs.

The simulated users of the system can be classified as:

- IT support;
- Researchers;
- Students;

Researchers are divided into groups, where each group has an allocation of resources, but individual researchers may have grants entitling them to additional resource usage.

Students are grouped by the curriculum they are enrolled in, and all students have a cap on the maximum usable resources, both cumulative and instantaneous, which may depend on their group.

The batch system shall match job requests to resources; there will be different classes of jobs for different uses, e.g. short, medium, long running, gpu,

interactive, etc. Typically, job classes/queues are distinguished not only on the running time but also on the amount of resources that can be requested.

The job queues are maintained in an order established by an auxiliary algorithm called a scheduler; the interface to the scheduler should be designed so that it would be easy to test different algorithms. Indeed, the overall performance of the computing system in actual operation strongly depends on the scheduler, therefore the evaluation of a given scheduler is one of the most important features of the simulation.

The proper design and implementation of an efficient scheduler is a task beyond the scope of the current module; hence, for the purposes of this assignment, it is sufficient to implement and interface a simple first-come first-served scheduler, provided the interfacing mechanism is designed to allow for later experimentation with other schedulers[1].

The policy constraints are:

- There are at least five different job queues:
  1. Short, interactive jobs that can take up to 2 nodes for no more than 1 hour; a certain subset (say 10%) of the machine must be reserved for this queue;
  2. Medium-sized jobs that can take up to 10% of the total number of cores, and can last up to 8 hours; another subset (say 30%) of the machine must be reserved for this queue
  3. Large jobs, that can take up to 16 hours and up to 50% of the total core count;
  4. GPU, for jobs requiring nodes equipped with them;
  5. Huge, active only from 0500pm Friday to 0900am Monday, where the jobs can potentially reserve the whole machine. During these times the other job queues do not serve requests.
- The cost of a node-hour has one value for all normal nodes, and a different, slightly higher value for GPU-enabled nodes;
- Each job will request a certain number of processor cores for a certain amount of time.

  The order in which the jobs are served is determined by the scheduler algorithm; as mentioned above, for the sake of simplicity you can assume a first-come first-served scheduling, with the proviso of easy maintenance/replacement;
- At the end of the week, there will be a cutoff time such that no new jobs will start if their estimated completion time will go beyond the end of the work week (thereby leaving the machine free for the weekend queue).

[1]Note that in a real world situation the system should be used "optimally", but different stakeholders may have different optimality criteria; for instance, users typically want to minimize waiting time, whereas administrators strive to maximize system utilization.

The machine can be assumed to have a constant operational cost per hour.

In the simulation program, users will be modeled as producers of requests, with each user having a certain budget; each user will produce requests up to his/her available budget. The exact resource requirements for each job will be created with the aid of a pseudo-random number generator. For simplicity, it is allowed (but not required) for each user to generate requests for only one job class. Within each class the amount of resources requested by each job will be modeled by a normal probability distribution; the workload request generation shall also take into account the job queue limits.

Each run of the simulator shall define a specific scenario comprising the set of simulated computer users, i.e. their number, distribution in classes and budgets. These parameters are chosen by the simulator users, i.e. the IT department staff that are investigating the policies for the new supercomputer.

During the course of the simulation, each simulated user will produce requests; the time between any two successive job requests shall be modeled by an exponential probability distribution, with parameters dependent on the class of user.

The output from the simulation should include

- The number of jobs processed in each queue (throughput) per week;
- The actual number of node-hours consumed by user jobs;
- The utilization ratio (number of node-hours consumed divided by number of node-hours available);
- The resulting price paid by the users;
- The average wait time in each queue;
- The average turnaround time ratio, i.e. the time from placing the job request to completion of the job divided by the actual runtime of the job;
- The economic balance of the centre, calculated by subtracting from the actual price the operating costs.

Assignment Requirements

The system shall be developed in either C++ or Python; it is strongly recommended that you reuse, to the extent possible, software components developed in other courses, provided you clearly identify in the accompanying and/or internal documentation which components have been reused "as is", which have been reused with modifications, and which have been written expressely for this assignment. You shall:

1. Choose appropriate models to represent the requirements (functional and non-functional);

2. Develop a test plan for the software. Explain what needs to be tested and how you will implement the tests. Ensure you have adequately considered both unit and integration testing, and explain your choice of test inputs; you can use any further validation testing you think appropriate;

3. Implement the software and run the tests from the plan, reporting their output;

4. Determine the test coverage that your test plan has achieved. Specify the percentage coverage of statement, decision and path coverage for each component, and thereby make a statement of the coverage across your whole application.

The code shall compile and execute on Crescent; for the C++ version you are free to choose between the Intel toolchain or the GNU toolchain. You are also free to develop on your own computer as long as the final code compiles and executes correctly on Crescent.

Assignment Deliverables

Ensure that all source code can be compiled and executed, by including the relevant make/project files

- A report (to be submitted on Turnitin) containing:
  - A description of the design employed;
  - A Software Requirements Document;
  - A Test Plan Document;
  - A discussion of test results and code coverage;

  The report shall be submitted individually by all mebers of a group; the report shall be mostly identical, but each member will add an individual section (at the end) describing the division of work and the member's individual contribution.

- Working Source Code with Tests (on BlackBoard, with makefiles as appropriate);

Marking Scheme

30 Software design and implementation: structure, internal documentation;

30 Requirements specification and project plan;

25 Test plan;

15 Test results and discussion, including code coverage.

Notes

You are free to make any reasonable assumption to supplement any missing or not fully specified requirements, but you have to state explicitly your additional assumptions in the documentation.

You are not required to try different queueing/pricing strategies, as long as the software is designed to allow such experimentation at a later time.

You will be evaluated on the quality of the software design, of the documentation and of the test plan.

You are encouraged to make use of standard documentation formats to structure your reports; the IEEE standard templates (among the most commonly used) are available in the support material on Blackboard.

Submission deadline:

Full time: 09:30 am, December 16th, 2019

Part time: 09:30 am, January 6th, 2020