**On the Management and Efficiency**

**of Cloud Based Services (236365)**

**Winter 2010-2011**

Distributed Oblivious Load Balancing Using Prioritized Job Replication Simulation

### Project Report

# Submitted by:

###### Asi Bross

###### Assaf Israel

###### Eli Nazarov

### Table of contents:

[1 Project Description 3](#_Toc286507325)

[1.1 Purpose 3](#_Toc286507326)

[1.2 Goals 3](#_Toc286507327)

[1.2.1 Simulator implementation: 3](#_Toc286507328)

[1.2.2 Load balancing with bounded queues: 3](#_Toc286507329)

[2 Detailed Description 4](#_Toc286507330)

[2.1 Class Diagram 4](#_Toc286507331)

[2.2 Configuration 4](#_Toc286507332)

[2.3 Event Generator 5](#_Toc286507333)

[2.4 Server 5](#_Toc286507334)

[2.5 Statistics Collector 5](#_Toc286507335)

[3 Results 6](#_Toc286507336)

# Project Description

## Purpose

The purpose of this project is to conduct a study regarding the behavior of a distributed oblivious load balancer using prioritized job replication with bounded queues. Implement the simulator and draw conclusions regarding the difference between the behavior of a system with infinite and finite queues.

## Goals

### Simulator implementation:

The simulator should be configurable using a configuration file (XML based). The configuration enables to trigger the queues, high priority (HP) and low priority (LP) bounds, the number of jobs, the number of servers and the load on the system. Also, it’ll be possible to extract information of the simulation in an easily manner into a CSV and/or XML files.

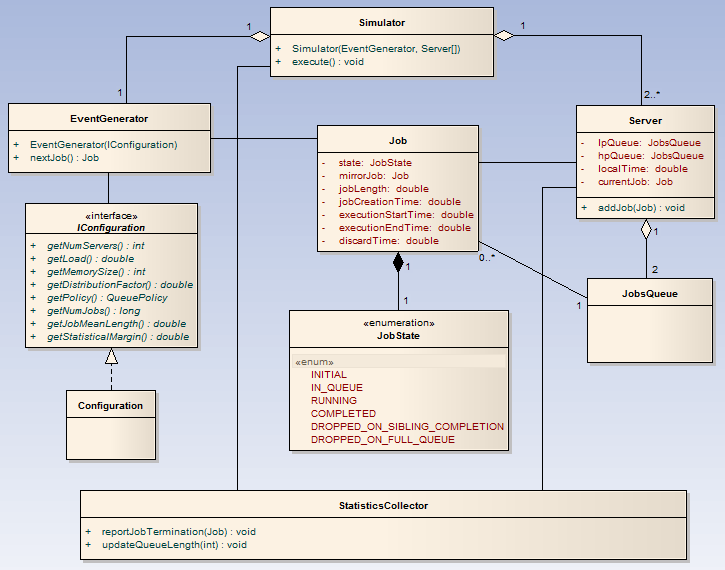
### Load balancing with bounded queues:

First, we’ll simulate different loads, with different number of servers, having infinite queues in order to investigate the scenario of M/M/2 and see if our results confirm with those of Amir Nahir study.

Simulate the systems behavior with bounded queues and different loads with different number of servers. Understand and present the key differences between the behavior of the system with infinite queues and a system with bounded queues.

# Detailed Description

## Class Diagram



## Configuration

The simulator receives only two arguments in the form of an input XML file and the results output file path. In the input file one can specify many different experiment configurations that will be executed and concluded in the output file. Each experiment configuration can modify the following experiment parameters:

* Number of servers
* Queue policy (finite/infinite) as well as the memory size (M) and the distribution factor (d) (relevant in finite policy only) which controls the distribution of memory between the two queues. In case of a finite queue policy the high priority queue capacity is , while the lower priority queue capacity is .
* Load
* Jobs count
* Jobs mean length
* Statistical margin – this indicates the fraction of initial and terminal jobs which will be discarded from the simulation statistics.

## Event Generator

According to the given configuration this element will create jobs in a poissonic rate and exponential distributed length. We use *Apache-Common-Math* library to produce randomized numbers according to an exponential distribution. The job length is exponentially generated according to the given job mean length, while the interval between two consecutive jobs is an exponential with the following average:



After deciding a job attributes the system chooses two different servers (in an uniform fashion), replicate the job and send it to each one with a different priority.

## Server

The server module responsibilities are to simulate the processing of jobs correctly, progress the time accordingly and alert the Statistics Collector that a job was finished with the correct execution attributes (i.e. execution start time, execution end time …).

There are multiple servers running in each simulation according to the experiment configuration and they signal each other through invocation of method calls.

Each server holds two job queues, one with high priority and the other with low priority, the server manages its own local time which is being updated either when a new job is added to one of the server's queues or through a signal that was sent from another server.

When a new job is added to the server, the server's local time is being updated to the job's creation time and all operation that should have been processed in that time interval are being handled accordingly, this includes signaling other servers.

The server alerts the Statistic Collector in the following execution's instrumental points.

* Job finished successfully
* Job dropped due to sibling's completion
* Job dropped since queue was full
* Jobs Queue size changed

## Statistics Collector

Statistics Collector (SC) is a module that is responsible for accumulating the statistics of the system.

Each server reports the termination of jobs and the changes in the length of the queues to a local SC. At the end of the simulation each SC reports its statistics to a global SC (GSC) that calculates the statistics of the entire system.

Thus, it is possible to receive detailed statistics of a simulation regarding each individual sever the whole system.

This module gives functionality of generating a report into a CSV and XML based files.

The statistics collector collects the following statistics:

HP Queue max length, LP Queue max length, HP Queue average length, LP Queue average length, HP Jobs average time in system, LP Jobs average time in system,

Also it provides statistics regarding the scenarios when a job, either LP or HP, is dropped due to sibling completion or full queue.

# Results

In order to validate our implementation against Amir’s study we’ve measured a series of experiments with infinite queues and measured average queue length and average jobs waiting time. As expected (and seen in the following figures) the queue length are exponential in regards to load and the in both queue types there’s a noticeable similarity.

Each measurement was taken as an average of a sequence of 100,000 jobs with a 10% statistical margin.

We’ve also looked for the load where the LP queues help the most. For this we’ve plotted the completion state percentage of LP jobs and HP jobs and looked for the load where the maximum number of LP jobs were completed. As we can see from the following figure this was achieved at ~0.75 load.

After we’ve validated our simulation performs as expected it was time to assess the finite queue policy. We wanted to evaluate how the load balancer will perform when coped with a serious load and less than sufficient resources (i.e. low memory capacity) for this we’ve looked at the average queues length when the system performs in infinite queue policy and 0.9 load. The average value of both queues was ~4, and so we’ve tested the finite policy system with a total memory capacity of 6 (which is 75% of the average total demand of 8). We’ve tracked after the same statistics when the memory distributed differently between the two queue types.

One can see that the average queue length and job waiting time is linear with queue size, however the system almost never utilize the entire memory available. The only exception to the linearity is when the system has only low priority queues. That can be explained by the fact that we’re always processing a low priority job (when available), much like the case where there is only enough capacity for high priority jobs.

In order to assess just how this limitation affects the completion state of jobs we’ve measured the percentage of jobs completion states under the same memory distribution. We should note that there are 5 different completion states tuples:

* HP Job completed & LP Job dropped-on-sibling-completion
* HP Job completed & LP Job dropped-on-full-queue
* HP Job dropped-on-sibling-completion & LP Job completed
* HP Job dropped-on-full-queue & LP Job completed
* HP Job dropped-on-full-queue & LP Job dropped-on-full-queue   
  (i.e. Job discarded completely)

We can see that the system is optimal (i.e. minimum percentage of discarded jobs) when there’s a small amount of memory dedicated for low priority queues, giving enough memory for high priority queues but letting the low priority queues help as much as possible (dark blue).

We’ve also repeated the experiment for higher load of 0.95 with similar results which confirm our conclusion.