Invited: Performance Evaluation of Heterogeneous Multi-Queues with Job Replication

Daniel A. Menascé and Noor Bajunaid
Department of Computer Science
George Mason University
4400 University Drive, Fairfax, VA 22030, USA
menasce@gmu.edu and nbajunai@masonlive.gmu.edu

***Abstract*—A system composed of n heterogenous servers receives jobs to be processed. A front-end dispatcher submits d ($d \leq n$) copies of each job to the queue of each of the d chosen servers. The copy that completes first causes the removal from the system of each of the other $d-1$ copies, which can be executing or in the queue for a server. This paper evaluates job execution time statistics as a function of (1) the average job arrival rate from a Poisson process, (2) the distribution of job service times, (3) the processing capacity of each server, (4) the job replication and dispatching policy. The paper derives analytical expressions for the average processing time of jobs that fail during execution and uses discrete event simulation to carry out experiments for analyzing a variety of statistics (e.g., response time, probability that jobs are cancelled while in the queue or killed while in execution, and server utilizations) related to a heterogeneous multi-server system under four different dispatching policies.**

I. Introduction

As indicated in [3], it may be advantageous to submit a job to multiple server queues at the same time when there is significant server side variability, which could be due to factors such as background load, network interrupts, garbage collection, and sotfware failures. By using redundancy, latency can be reduced and a job is considered to be complete when it finishes ahead of all its replicas, which are *cancelled* if they are still waiting to start processing or are *killed* if they have already started to be processed. Previous theoretical queuing research on redundant queues used a simplifying independence assumption that the service time of all job replicas are drawn from identical and independent random variables (see e.g., [4], [6], [8]). However, all the replicas are identical in terms of their computing and I/O demands and therefore one cannot assume that the service times of all replicas of a job are *independently* drawn from the same distribution at each server.

Our study considers that the servers in a cluster are basically heterogeneous in terms of capacity even if they start as identical. The reasons may include: (1) servers that age or fail tend to be replaced by newer and faster ones; (2) the state of seconday storage devices among the servers of a cluster diverges in different ways over time leading to different I/O times; (3) server virtualization leads to different server capacities due to the co-existence of various virtual machines in a single physical server; and (4) the background load of OS services varies from server to server.

We also consider here that jobs may fail during their execution due to a variety of factors, including memory leaks and misconfigurations, in which case jobs have to be restarted. Jobs may fail several times before they succeed. Therefore, a job's processing time is elongated by the amount of computation wasted due to failures.

This paper examines and compares four different dispatching policies. Two of them (Random and Redundant-to-Idle-Queue) use none or very little information about the servers to make dispatching decisions. The other two (Average Queue Length and Queue Cancellation) use server statistics to estimate the response time of a job at each server before deciding where to send a job and its replicas. These two policies are more intrusive than the first two and may not be easy or practical to implement in some environments.

The contributions of this paper include: (1) A formal statement of the problem of dispatching replicas of a job to heterogeneous servers. (2) The derivation of an analytical expression for the average job processing time in the presence of job failures. (3) An extensive simulation study of the variation of the response time and other related metrics as a function of the number of job replicas for several dispatching policies.

The rest of the paper is organized as follows. Section II formally describes the problem. Section III presents the assumptions used in the paper. Section IV introduces the notation used here. Section V discusses the motivation for dispatching multiple copies of a job. The next section provides a derivation of the average number of failures and wasted time due to job restarts. Section VII describes the replication dispatching policies considered in this paper and Section VIII describes the simulation experiments and their results. Finally, Section IX provides some concluding remarks and discusses future work.

II. Problem Description

Consider a server cluster composed of n servers and a *front-end job dispatcher* (heretofore referred as dispatcher). Jobs arrive to the dispatcher, which creates d ($1 \leq d \leq n$) replicas of the job and submits them to server queues chosen according to a *dispatching*

policy. The set of replicas of a job is called a *job group*. The first job of a job group to finish causes all other $d-1$ replicas to be cancelled or killed.

Figure 1 shows an example of the system we consider in this paper. There are four servers and the dispatcher is shown receiving three jobs (J_1, J_2 and J_3) in this order. Two copies are made of each job. The two copies of job J_1 are submitted to servers 1 and 3, the copies of job J_2 are submitted to servers 3 and 4, and the two copies of job J_3 to servers 2 and 4. If the copy of job J_1 finishes in server 1 before its copy in server 3, that copy is deletd whether it is still in server 3's queue or being processed by server 3.

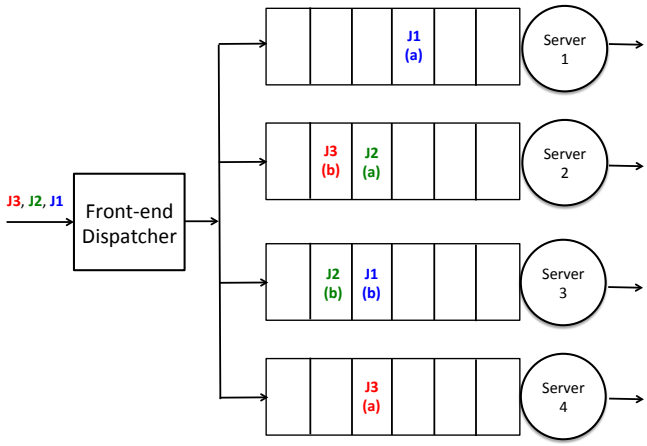

Fig. 1: Multiple Queues with Job Replication.

We consider, for the reasons below, that servers are heterogeneous. In practice, it is more common for servers in a cluster to have different effective capacities for the following reasons:

- As servers age or fail, they are replaced by newer ones with typically different characteristics (e.g., different clock frequencies, different cache sizes, and even different hardware architectures). Thus, even a cluster of identical servers gradually morphs over time into a cluster of heterogeneous servers.
- The performance of two identical servers tends to diverge over time due to the state of their secondary storage.
- When servers are virtual servers running on physical servers, their effective capacity is affected by the load of other virtual machines running on the same physical box [10]. Thus, in this case, a server's capacity tends to change over time.
- The background load of OS services varies over time from server to server and uses resources required by the jobs.

III. Assumptions

We consider the following assumptions:

- A1: The cancellation time of a job in the queue is negligible because it involves trivial operations in the data structure that represents the queue.
- A2: The time to kill a job (i.e., removal of a job when it is being processed) is considered to be negligible because it involves process context switching, which is typically very small compared with a job's processing time. It should be noted though that the penalty for killing a job is that a server wastes time on that job, delaying other jobs that arrived after the killed job. We assume that the dispatcher can keep track of the time spent processing by a job that is killed at any server (see also assumption A7). Therefore, the dispatcher can compute the average time a job wastes being served before it is killed by dividing the sum of the wasted times by the number of killed jobs at each server.
- A3: Jobs arrive to the dispatcher according to a Poisson process. This is a reasonable assumption when jobs are generated from a large number of independent sources.
- A4: The dispatcher knows the distribution of the number of job operations as well as the capacity of each of the n servers. This means it knows the distribution of job's service times. However, the dispatcher does not know the actual service time of individual jobs because the actual execution time is usually data-dependent.
- A5: Servers use a FCFS discipline to serve jobs in their queues.
- A6: The dispatcher knows the status of each queue (i.e., the list of jobs in each queue) and the id of the job being served but it does not know the execution times of the jobs in the server queues nor the execution time of the jobs being processed.
- A7: The dispatcher is notified when a job is cancelled or killed at any server. Therefore, the dispatcher can keep a running estimate of the probability that jobs are cancelled or killed at any server.

IV. Notation

Consider the following notation:

- n: number of servers
- d ($d \leq n$): number of replicas generated for a job.
- $\tilde{O}$: r.v. that indicates the number of operations of a job.
- C_i ($i = 1, \cdots, n$): processing capacity of server i, in operations/sec. We assume without loss of generality that $C_1 \leq C_2 \leq \cdots \leq C_n$.
- $\tilde{S}_i$: r.v. that indicates the processing time of jobs at server i; thus, $\tilde{S}_i = \tilde{O}/C_i$.
- λ: average arrival rate of jobs at the dispatcher.
- λ_i: average arrival rate of jobs at server i.
- $f(\mathcal{P}, i)$: fraction of jobs received by the dispatcher that are sent to server i under replication policy $\mathcal{P}$.
- R_i: average response time of jobs at server i.
- $P_{\text{cancel},i}$: probability that jobs are cancelled at server i, computed as the ratio of the number of jobs cancelled at the server over the total number of jobs received by the server.

- $P_{\text{kill},i}$: probability that jobs are killed at each server, computed as the ratio of the number of jobs killed at the server over the total number of jobs received by the server.
- $P_{\text{complete},i}$: probability that jobs that arrive at server i complete their processing at that server, i.e., they are neither cancelled nor killed. Thus, $P_{\text{complete},i} = 1 - (P_{\text{cancel},i} + P_{\text{kill},i})$.
- $E[\text{Wasted}_{\text{kill},i}]$: average processing (not waiting) time spent by killed jobs at server i. This value is computed by the dispatcher (see Assumption A2).
- $\tilde{F}_i$: r.v. that represents the failure instant of a job running at server i after its last start or restart at that server.
- $f_{\tilde{F}_i}(x)$: pdf of $\tilde{F}_i$.
- q_i: probability that a job fails while executing at server i
- $\tilde{W}_i$: r.v. that denotes the amount of wasted computation time at server i per failure of a job at server i
- $\tilde{P}_i$: r.v. that indicates the total processing time of a job at server i including successful processing $\tilde{S}_i$ plus the total wasted time due to all failures of the job.
- $U_{u,i}$: useful utilization of server i defined as the ratio of the time the server was busy processing requests that were not killed at the server divided by the total simulation time.
- $U_{w,i}$: wasted utilization of server i defined as the the ratio of the time the server was busy processing jobs killed at the server divided by the total simulation time.

V. Motivation

As a motivation for our work, we show in Fig. 2 a graph of average response time versus the number of replicas d for a case of 5 servers. The arrival rate of requests to the dispatcher is 0.072 req/sec, the number of operations per job is geometrically distributed with parameter $p = 0.01$ and all servers have the same nominal capacity of 10 operations/sec. All servers have Poisson failure models with different failure rates: $\beta_1 = 0.01$, $\beta_2 = 0.02$, $\beta_3 = 0.04$, $\beta_4 = 0.05$, and $\beta_5 = 0.05$. In this example, the dispatcher randomly selects d out of the 5 servers to send replicas of a job. The picture shows that for the data above, the minimum response time occurs when the number of replicas is equal to 3. In other words, dispatching more than one replica of the job reduces the response time to a point but when too many replicas are submitted, the response time increases due to excessive wasted time by killed jobs.

Figure 3 shows the relationship between cancelled, killed, and completed requests as they flow through the dispatcher, go through one or more servers, and leave. The overall arrival rate of requests to the dispatcher is λ requests/sec. A fraction $f(\mathcal{P}, i)$ of these requests is routed to server i according to the dispatching policy $\mathcal{P}$.

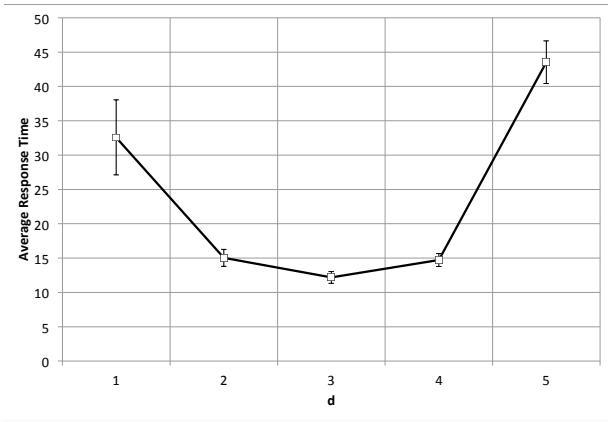

Fig. 2: Average response time with 95% confidence intervals as a function of the number of replicas d.

So, the arrival rate of request to server i is

$$\lambda_i = \lambda \times f(\mathcal{P}, i). \tag{1}$$

Then, the rate at which requests complete from server i, i.e., the throughput X_i, is

$$X_i = \lambda \times f(\mathcal{P}, i) \times P_{\text{complete},i} \tag{2}$$

We can then compute the useful utilization, $U_{u,i}$ of each server i as

$$U_{u,i} = \lambda \times f(\mathcal{P}, i) \times P_{\text{complete},i} \times E[\tilde{P}_i] \tag{3}$$

An upper bound for the wasted utilization, $U_{w,i}$, of server i due to job killings is

$$U_{w,i} < \lambda \times f(\mathcal{P}, i) \times P_{\text{kill},i} \times E[\tilde{P}_i] \tag{4}$$

because the average time spent by the server on a killed job is less than $E[\tilde{P}_i]$.

As another example, Figs. 4-6 show graphs depicting $P_{\text{cancel}}, P_{\text{kill}}$ and P_{complete} for each of nine servers for random dispatching to d servers for $d = 2, 5$ and 9, respectively. The capacity of the nine servers is given by $C_i = 2\ i$ operations per second. So, the capacity ranges from 2 to 18 operations per second with server 1 being the slowest and server 9 the fastest. The arrival rate of requests is 0.02 requests/sec and the number of operations per request is distributed according to the binomial Bin (150, 0.5).

As a general observation with respect to Figures 4-6, we can see that the cancel probability is very low but the kill probability is high and close to 1 for the slower servers and decreases for the faster servers. The completion probability increases with the speed of the servers, i.e., faster servers are more likely to complete a job.

If the actual execution times of the jobs were known by the dispatcher, it could submit just one copy of the job to the shortest queue. However, according to assumptions A4 and A6, the dispatcher does not know the actual execution times, just the service time distribu-

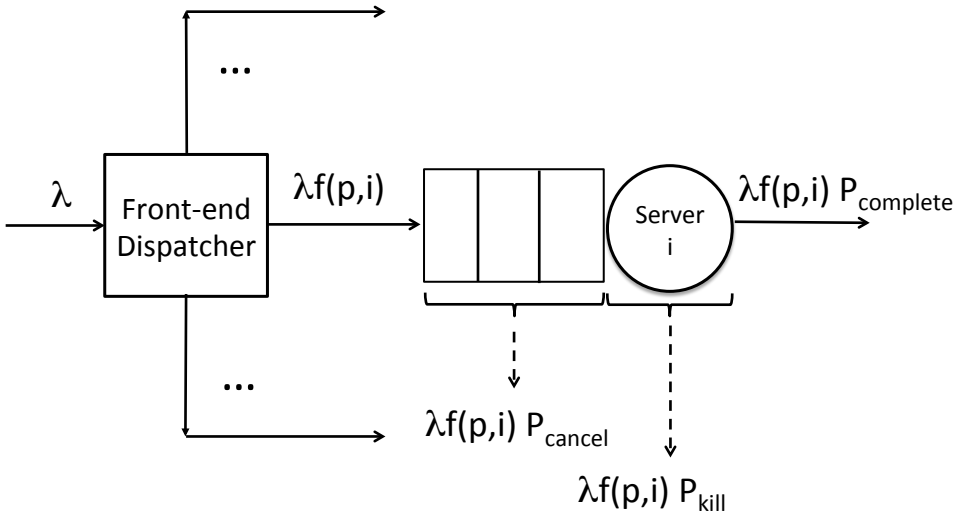

Fig. 3: Flow of Requests.

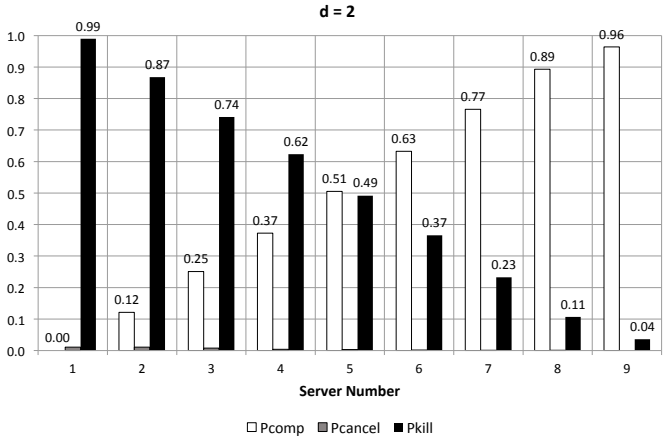

Fig. 4: $P_{\text{cancel}}, P_{\text{kill}}$ and P_{complete} for $d = 2$ and random dispatch.

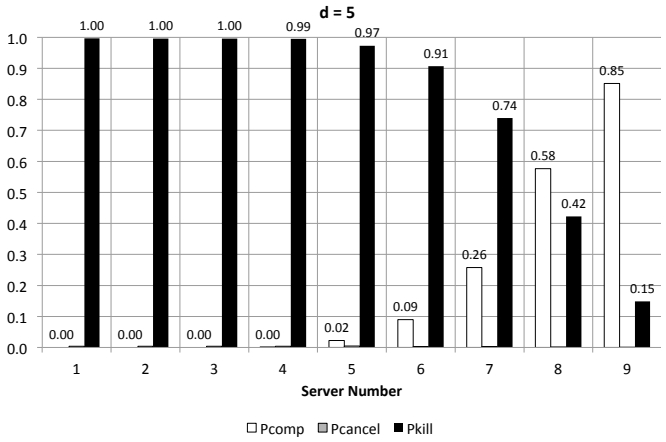

Fig. 5: $P_{\text{cancel}}, P_{\text{kill}}$ and P_{complete} for $d = 5$ and random dispatch.

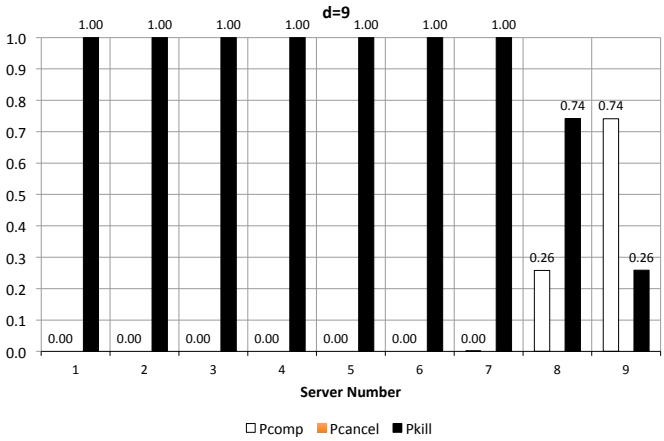

Fig. 6: $P_{\text{cancel}}, P_{\text{kill}}$ and P_{complete} for $d = 9$ and random dispatch.

tions. Therefore, submitting more than one copy gives a job a better chance of joining a queue where the job will be served with a lower response time. Then, it would appear that n copies should be made of each arriving job and they should be submitted to all servers. However, according to assumption A2, jobs in a queue are delayed by the partial execution time of jobs that are killed. So, the more copies are generated the more we increase the response time in each queue. Therefore, there is an optimal balance between these two effects (i.e., increasing the chance of joining a queue with the smallest waiting time and joining a queue with significant delays due to jobs that will end up being killed).

VI. Failures During Job Processing

It is not uncommon for a job's execution to stall because of a variety of reasons such as hardware faults, memory leaks, or misconfigurations. In fact, Hadoop, an open source implementation of Google's MapReduce,

creates a copy of a task on another node if the original one is performing poorly. This is called speculative execution and it has been noted to improve job response times by 44% [2], [10]. Therefore, we assume that jobs can fail during execution and that they are restarted on the same node without any significant delay. We do not consider hardware failures here, just software-related failures.

Consider that a job executes k operations. So, its execution time without failures at server i is $S = k/C_i$. The probability $q_i(S)$ that the job fails before completing its execution, conditioned on the value of S, is the probability that the failure instant occurs before the end of the execution. So,

$$q_i(S) = \int_0^{S^-} f_{\tilde{F}_i}(x)\, dx \tag{5}$$

The probability of exactly j failures experienced by the job with execution time S executing at server i before it succeeds in completing its execution is $q_i(S)^j(1-q_i(S))$ assuming independence among failures. For example, if the failure model for node i is Poisson, i.e., $\tilde{F}_i$ is exponentially distributed with parameter β_i, then $q_i(S) = 1 - e^{-\beta_i S}$.

The average number of failures $NF_i(S)$ before a job is able to complete at server i is

$$NF_i(S) = \sum_{j=0}^{\infty} j \times q_i(S)^j(1-q_i(S)) = \frac{q_i(S)}{1-q_i(S)}. \tag{6}$$

The above expressions are conditioned on the service time being equal to S. We can uncondition the expression for NF_i as follows:

$$\begin{aligned} NF_i &= \sum_{k=1}^{\infty} \frac{q_i(k/Ci)}{1-q_i(k/Ci)} Pr[\tilde{S}_i = k/Ci] \\ &= \sum_{k=1}^{\infty} \frac{q_i(k/C_i)}{1-q_i(k/Ci)} Pr[\tilde{O} = k] \end{aligned} \tag{7}$$

because $Pr[\tilde{S}_i = k/C_i] = Pr[\tilde{O} = k]$.

Assume as an example that the r.v. $\tilde{O}$ is geometrically distributed with parameter p. Then $Pr[\tilde{O} = k] = (1-p)^{k-1}p$. Also assume as above, for the sake of example, that the failure model is Poisson. Then, $q_i(k/C_i) = 1 - e^{-\beta_i k/C_i}$ and

$$\begin{aligned} NF_i &= \sum_{k=1}^{\infty} \frac{1 - e^{-\beta_i k/C_i}}{e^{-\beta_i k/C_i}} (1-p)^{k-1}p \\ &= \frac{p}{e^{-\beta_i/C_i} - 1 + p} - 1 \end{aligned} \tag{8}$$

if $[(1-p)/e^{-\beta_i/C_i}] < 1$. Note that Eq. (8) yields $NF_i = 0$, as expected, when there are no failures, i.e., $\beta_i = 0$.

The average value, $E[\tilde{W}_i \mid \tilde{S}_i = S]$, of $\tilde{W}_i$ given that $\tilde{S}_i = S$ is the average value of $\tilde{F}_i$ given that $\tilde{F}_i < S$ because when a failure occurs before a job completes its S seconds of computation, the computation up to the failure instant is wasted. The expression for $E[\tilde{W}_i \mid \tilde{S}_i = S]$ follows from the definition of conditional probability:

$$\begin{aligned} E[\tilde{W}_i \mid \tilde{S}_i = S] &= E[\tilde{F}_i \mid \tilde{F}_i < S] \\ &= \frac{\int_{x=0}^{S^-} x.f_{\tilde{F}_i}(x)dx}{\int_{x=0}^{S^-} f_{\tilde{F}_i}(x)dx}. \end{aligned} \tag{9}$$

We can then uncondition Equation (9) as follows

$$E[\tilde{W}_i] = \sum_{k=1}^{\infty} E[\tilde{W}_i \mid S = k/C_i] Pr[\tilde{O} = k]. \tag{10}$$

For example, consider as before that $\tilde{F}_i$ is exponentially distributed and $\tilde{O}$ is geometrically distributed. Then, $E[\tilde{W}_i \mid \tilde{S} = S]$ can be computed as

$$E[\tilde{W}_i \mid \tilde{S}] = S] = \frac{\int_{x=0}^{S^-} x.\beta_i e^{-\beta_i x}dx}{\int_{x=0}^{S^-} \beta_i e^{-\beta_i x}dx} \tag{11}$$

$$= \frac{\frac{1}{\beta_i} - \left(\frac{1}{\beta_i} + S\right) e^{-\beta_i S}}{1 - e^{-\beta_i S}}. \tag{12}$$

Unconditioning on $S = k/C_i$ gives us

$$E[\tilde{W}_i] = \sum_{k=1}^{\infty} \frac{\frac{1}{\beta_i} - \left(\frac{1}{\beta_i} + (k/C_i)\right) e^{-\beta_i k/C_i}}{1 - e^{-\beta_i k/C_i}} .(1-p)^{k-1}p \tag{13}$$

The infinite summation in Equation 13 can be computed algorithmically (see Algorithm 1) by observing that as $k \to \infty$, $e^{-\beta_i k/C_i} \to 0$, and using L'Hospital's Rule, $ke^{-\beta_i k/C_i} \to 0$. Note also that $e^{-\beta_i k/C_i} \to 0$ much faster than $(1-p)^{k-1}$. Thus, for sufficiently large values of k (say $k = k^*$), the term inside the summation becomes $\approx (1/\beta_i)(1-p)^{k-1}p$ and the summation from $k = k^*$ to ∞ converges to $(1-p)^{k^*-1}/\beta_i$. Therefore, the summation in Equation (13) can be computed by adding the k-th term starting at $k = 1$ until the k-th term is close enough (within a tolerance) to $(1/\beta_i)(1-p)^{k-1}p$ (lines 3-6 of the algorithm). We then need to add $(1-p)^{k^*-1}/\beta_i$ to the already accumulated values (line 7 of the algorithm).

Algorithm 1 Algorithmic Computation of Eq. (13)

Input β_i, p, C_i
Output $E[\tilde{W}_i]$

1: Let $T(k) = \frac{\frac{1}{\beta_i} - \left(\frac{1}{\beta_i} + (k/C_i)\right) e^{-\beta_i k/C_i}}{1 - e^{-\beta_i k/C_i}} .(1-p)^{k-1}p$
2: $k \leftarrow 1$; SUM $\leftarrow 0$;
3: **repeat**
4: SUM $\leftarrow$ SUM $+ T(k)$
5: $k \leftarrow k + 1$
6: **until** $T(k) \approx (1/\beta_i)(1-p)^{k-1}p$
7: $E[\tilde{W}_i] \leftarrow$ SUM $+ (1-p)^{k-1}/\beta_i$

We can now combine Eqs. (8) and (13) to compute the average job processing time $E[\tilde{P}_i]$ at server i, considering job failures, as

$$E[\tilde{P}_i] = E[\tilde{S}_i] + NF_i \times E[\tilde{W}_i]. \tag{14}$$

VII. Replication Policies

A replication policy determines how many copies of a job should be generated and to which servers they should be submitted. We consider the following policies for selecting queue placement for job J. Some of these policies are oblivious to the state of the servers while others use some information about the state of the servers. Clearly, the policies that use more information from the servers are expected to perform better. However, in practice, it may not be feasible for the servers to share significant state information with the dispatcher, especially when the number of servers is very large as is the case with many warehouse-scale data centers [1].

- *AverageQLength (AQL)*: Compute for each server i an estimate R_i of the response time of an arriving job J at server i as

$$R_i \approx E[\tilde{P}_i] + QL_i.E[\tilde{P}_i] \tag{15}$$

  where $E[\tilde{P}_i]$ is the average processing time of jobs at server i computed using Eq (14) and QL_i is the queue length at server i including the job being served. QL_i is known to the dispatcher according to Assumption A6. Note that R_i is a conservative estimate because it is possible that one or more of the jobs found in the queue by job J will be cancelled or killed. The AQL policy then selects the d servers with the smallest values of R_i to submit job J.
- *QCancellation (QC)*: As indicated in Assumption A7, the dispatcher knows when jobs are cancelled or killed at any server. Therefore, the dispatcher can keep a running estimate of the values of $P_{\text{complete},i}$ and $P_{\text{kill},i}$ for each server by dividing, respectively, the number of jobs completed at the server by the number of jobs submitted to the server and the number of jobs killed at the server by the number of jobs submitted to the server. Thus, this policy is similar to the *AverageQLength* policy but takes into account the estimated probability that some of the jobs are killed or cancelled. Therefore, the dispatcher computes for each server i the following estimate for the average response time R_i as

$$\begin{aligned} R_i \quad \approx \quad & E[\tilde{P}_i] + \\ & QL_i\ [E[\tilde{P}_i].P_{\text{complete},i} + \\ & \qquad E[\text{Wasted}_{\text{kill},i}].P_{\text{kill},i}] \end{aligned} \tag{16}$$

  and submits a replica of job J to the d servers with the smallest values of R_i.
- *Redundant-to-Idle-Queue (RIQ)*: This dispatching policy was introduced in [3]. Under this policy, the dispatcher randomly choses d servers and queries them to check if they are idle or busy. If all d servers are busy, job J joins the queue of one of the d servers at random. However, if $0 < i \leq d$ queried servers are idle, job J is submitted to all i idle servers.
- *Random*: This policy randomly selects d different queues to submit job J. This policy, also referred to as Redundant-d in [3], is oblivious to the state of the queues. Note that the RIQ and the Random policies are identical when $d = 1$. Also, at light loads, RIQ and Random tend to behave similarly because there is a very high probability that all chosen servers are idle.

VIII. Simulation Study Results

This section describes the simulator and the setup for the simulation experiments. Figure 7 depicts the architecture of the simulator and its main components. The simulator was written in Java and considers jobs

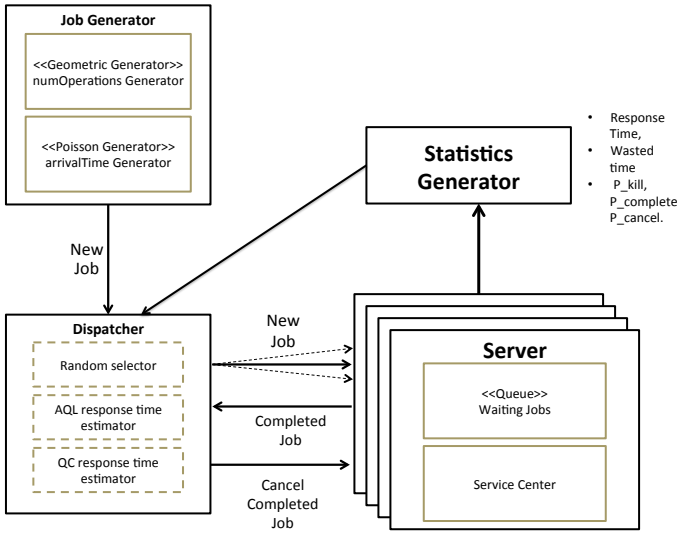

Fig. 7: Block Diagram of the Simulator.

that arrive to the dispatcher from a Poisson process with $\lambda = 7$ and 14 jobs/sec. The number of operations per job is distributed according to a geometric distribution with parameter $p = 0.01$ in our experiments. Other distributions could be used though. Thus, $Pr[\tilde{O} = k] = (1-p)^{k-1}p = 0.99^{k-1} \times 0.01$ for $k = 1, \cdots$. We used Poisson failures throughout our simulation studies. All of our simulation experiments consider 100 servers (i.e., $n = 100$) clustered into four groups: cluster 1 (servers 1-25), cluster 2 (servers 26-50), cluster 3 (servers 51-75), and cluster 4 (servers 76-100). Remember that lowered numbered servers have lower nominal capacity. Table I shows the profiles of the four clusters. The table shows the average, standard deviation, and coefficient of variation (COV) of the server capacities in each cluster as well as these statistics for the failure rates. The COVs for the server capacity are smaller than for the failure rates.

Our simulator computes the following metrics for each server and computes averages per cluster: average response time R_i, job cancellation probability

TABLE I: Profile of the four server clusters.

		Capacity (in ops/sec)			Failure Rate (in failures/sec)		
Cluster id	Range	mean	stdev	COV	mean	stdev	COV
1	1-25	67.0	9.7	0.14	0.014	0.008	0.57
2	26-50	99.6	7.7	0.08	0.032	0.014	0.43
3	51-75	135.5	14.2	0.10	0.040	0.018	0.45
4	76-100	175.8	11.6	0.07	0.054	0.023	0.42

P^i_{cancel}, job killing probability P^i_{kill}, job completion probability P^i_{complete}, average useful utilization $U_{u,i}$, and average wasted utilization $U_{w,i}$. Note that the simulator learns, over time, estimates of P^i_{complete} and P^i_{kill} for each server.

We carried out two types of simulation experiments: (a) average experiments: study the average values of the metrics above and their corresponding 95% confidence intervals obtained from 30 simulation runs with 3,000 job completions in each and (b) time-progression experiments: study the evolution of the just described metrics over time. Results are plotted for every 50 job completions. Both types of studies show results for dispatching policies AQL, QC, RIQ, and Random for different values of the replication factor d. We used the same set of seeds (e.g., for job arrivals, job failures, number of operations per job) for each simulation run. That is, we used 30 seeds through our runs. Each seed is used to run simulations to cover all the combinations of the dispatching policies and the value of the replication factor d. Thus, we are comparing the policies and d values under the same sequence of jobs and under the same failures.

A. Average Simulation Results

The results shown in this section depict a variety of average and 95% intervals for the metrics described at the top of this section. We prepared three sets of graphs to show the average results after the simulation converged. For convergence, we let 3000 jobs complete in the simulation and then collect statistics from the completion of the next 3000 jobs. Figure 8 shows six different charts numbered 1 through 6 from left to right and then top to bottom as in the previous subsection.

The response time charts are normalized by the number of operations for each job and then averaged for all the jobs completed at the system or at a server cluster.

- Fig. 8 (1): depicts the variation of the average response time from all the jobs at the system, vs. the replication factor d when the arrival rate $\lambda = 7$ jobs/sec, for the four dispatching policies. The AQL and QC dispatching policies produce better response times for smaller values of d, while the response time from the Random and RIQ policies yield better results as we increase d. The best results overall are obtained for $d = 1$ and policies AQL and QC, which are statistically equivalent at the 95% confidence level. It should be noted however that these policies require detailed knowledge of the operation of the servers.
- Fig. 8 (2): this figure is similar to Fig. 8 (1) but considers a job arrival rate twice as big (i.e., $\lambda = 14$ jobs/sec).These two charts are similar but the response times are higher overall for all policies as expected. It is interesting to note that QC is better than AQL for $d > 10$ because the queue prediction of QC is more accurate.
- Fig. 8 (3): depicts the variation of the average response time for jobs completed at each cluster, vs. the replication factor d when $\lambda = 14$ jobs/sec and for the Random dispatching policy. Under this policy, all the servers have a chance to receive an arriving job, and when $d = 1$ and $d = 2$, servers in clusters 1 and 2 have a better chance of completing their jobs. As expected, the average response time at cluster i is better than the average response time at cluster $j \; \forall \; i > j$ because servers in cluster i are faster, on average, than servers on cluster j. It can also be seen that, for all four clusters, the average response time tends to improve as we increase d to 4 or 5, and then it worsens for larger values of d. Increasing d to 4 or 5 increases the chance of arriving jobs being dispatched and completed at faster servers. However, for larger values of d, the contention at servers increases and thus, jobs will spend more time waiting for other jobs that most probably will be killed later.
- Fig. 8 (4): this figure is similar to Fig. 8 (3) except that it uses the RIQ policy. For smaller values of d, the results from the RIQ policy follow the same pattern as the Random policy because these policies behave similarly for small values of d. However, the RIQ policy tends to give better results for larger d values. This is because the RIQ policy does not increase contention as the Random policy because it dispatches jobs to idle servers only or to one server only in case no idle server is selected initially.
- Fig. 8 (5): this figure is also similar to Fig. 8 (3) except that it uses the AQL policy. Under this policy, an arriving job will be dispatched to the servers with d shortest estimated response time. For $d = 1$, we see results for cluster 4 only, which means that all the jobs were dispatched to cluster 4 servers

because no cancelation or killing will occur for $d = 1$. For larger values of d, we see other clusters completing some jobs. This happens because as we increase d, we increase the contention among cluster 4 servers and therefore, servers in other clusters have a better chance of receiving and completing some of the jobs. All clusters experience longer response times for larger values of d. Under AQL, only 5 out of 30 simulation runs showed a response time for cluster 1 and $d = 4$, and 27 runs showed a response time when $d = 5$. In addition, only 2 out of 30 runs showed a result from cluster 2 for $d = 2$.

- Fig. 8 (6): this figure is also similar to Fig. 8 (3) but uses the QC dispatching policy. The results show the same pattern as the AQL results, except that we can see results for cluster 1 only for $d \geq 4$. In fact, out of 30 simulation runs, only 2 showed a response time from cluster 1 when $d = 4$, and 23 runs showed a response time for cluster 1 when $d = 5$.

In general, the charts in Fig. 8 show that the Random and RIQ policies give better response time with larger values of d, while AQL and QC give better response time for smaller values of d. In fact, under the AQL and QC dispatching policies, redundancy does not improve response time. However, these policies will always use the same set of servers. Random and RIQ require none (in the case of Random) or little (in the case of RIQ) knowledge of the status of the servers. AQL and QC are more intrusive and require the dispatcher and servers to share a lot more knowledge.

The second set of charts shows $P_{\text{complete},i}$ and $P_{\text{kill},i}$ for all four policies, $d = 1, 2, 3, 4, 5, 10$, and 15 and $\lambda = 14$ requests/sec.

Figure 9 shows eight different graphs numbered 1 through 8 from left to right and then top to bottom as in the previous subsection. We describe now each of the eight graphs and the observations we derive from them.

- Fig. 9 (1): depicts the variation of $P_{\text{complete},i}$ vs. d when the arrival rate $\lambda = 14$ jobs/sec under the Random dispatching policy. It is expected that $P_{\text{complete},i}$ decreases as d increases because more servers receive replicas of the same job. However, slower servers experience a faster decrease in $P_{\text{complete},i}$ while the decrease in cluster 4 is almost linear. This is because as we increase d, the competition for job completion tends to be between cluster 4 servers. So, most of the completions will still be seen from servers at this cluster.
- Fig. 9 (2): depicts the variation of $P_{\text{kill},i}$ vs. d when the arrival rate $\lambda = 14$ jobs/sec under the Random dispatching policy. We notice that this probability initially increases with d and then starts to decrease. The value of d at which this happens is not the same for each cluster. For example, this happens for $d = 5$ for cluster 1, for $d = 10$ for cluster 2, and for $d = 15$ for cluster 3. The figure does not show results for $d > 15$ but $P_{\text{kill},i}$ shows signs of decreasing for $d = 15$ for cluster 4. The explanation for this behavior is that as d increases, the Random policy will distribute jobs into more clusters in a uniform way. Therefore, more jobs tend to be killed by servers at higher capacity clusters and $P_{\text{kill},i}$ increases. However, as d goes above a threshold, the queue size increases at all servers and higher capacity servers will take longer to complete jobs and therefore will provide more chance for jobs not to be killed at other servers.
- Fig. 9 (3): this chart is similar to Fig. 9 (1) but uses the RIQ dispatching policy. The chart follows the same pattern as Fig. 9 (1), but with better overall completion probability. This happens because jobs will be dispatched to an idle server most of the time, and when a job is dispatched to one server, this server is guaranteed to complete the job.
- Fig. 9 (4): this chart is similar to Fig. 9 (2) but uses the RIQ dispatching policy. Under this policy, $P_{\text{kill},i}$ always increases with d. For clusters 1 and 2, $P_{\text{kill},i}$ becomes very high for $d = 10$ and $d = 15$. As d increases, the servers that are idle will most probably be idle due to constant killing and canceling and not due to not receiving jobs. So, when the same set of servers are found idle and selected by RIQ, $d - 1$ of these will kill their job, thus increasing the overall $P_{\text{kill},i}$.
- Fig. 9 (5): this chart is similar to Fig. 9 (1) but uses the AQL dispatching policy. The high value of $P_{\text{complete},i}$ for cluster 1 servers under lower d values is due to not receiving any jobs and thus not changing their initial value of $P_{\text{complete},i}$, which is 1. The decrease in $P_{\text{complete},i}$ for clusters 3 and 4 is mostly attributed to competition within the servers at these clusters.
- Fig. 9 (6): this chart is similar to Fig. 9 (2) but uses the AQL dispatching policy. The killing probability increases then decreases with d for all clusters. This is because servers start with $P_{\text{kill},i} = 0$, and if they do not receive any jobs, they will keep this probability. For larger values of d, faster servers tend to have longer queues, preventing them from receiving more jobs until their queues become shorter. So, many servers on these clusters will not receive any jobs for larger values of d, and thus their $P_{\text{kill},i}$ will be 0, which will lower the overall average $P_{\text{kill},i}$ from all the servers in the cluster.
- Fig. 9 (7): this chart is similar to Fig. 9 (1) but uses the QC dispatching policy and follows the same pattern as Fig. 9 (5) because the AQL and QC policies work under the same principle of estimating the queue size.
- Fig. 9 (8): this chart is similar to Fig. 9 (2) but uses the QC dispatching policy and follows the same pattern as Fig. 9 (6) for the reasons explained

Fig. 8: Results for Average Response Times.

above.

The third set of results shows four graphs for the useful and wasted utilization per cluster under the different policies for $d = 1, 2, 3, 4, 5, 10$, and 15 and $\lambda = 14$. Figure 10 shows four different graphs (one per cluster) numbered 1 through 4 from left to right and then top to bottom as in the previous subsection. Each graph displays four stacked bars (one per policy) for each value of d. The bottom part of each stacked bar represents useful utilization and the top part represents wasted utilzation. We describe now each of the four graphs and the observations we derive from them.

- Fig. 10 (1): depicts the variation of average server utilization for cluster 1 servers vs. d for various policies. Under Random and RIQ dispatching, cluster 1 servers will always be utilized, however, it is mostly a wasted utilization where all the work is wasted because of job killing. Under AQL and QC policies, the servers will only be used for larger values of d and it is mostly a wasted utilization.
- Fig. 10 (2): depicts the variation of average server utilization for cluster 2 servers vs. d for various policies. Similarly to cluster 1, the servers will be used under Random and RIQ policies for all values of d, but as d increases, most of the work will be wasted. Under AQL and QC policies, the servers will be utilized for $d \leq 4$. Again, most of this is wasted work.
- Fig. 10 (3): depicts the variation of average server utilization for cluster 3 servers vs. d for various

Fig. 9: Results for Average $P_{\text{complete},i}$ and $P_{\text{kill},i}$.

policies. Under Random and RIQ policies, the useful utilization decreases and the wasted utilization increases as d increases. However, under AQL and QC policies, the useful utilization is roughly the same for $d \leq 4$, but the wasted utilization increases.

- Fig. 10 (4): depicts the variation of average server utilization for cluster 4 servers vs. d for various policies. The servers in cluster 4 have the best useful utilization among all other clusters under all policies. This is expected as they have a better chance of completing the jobs due to their higher capacities. They even have better useful utilization under Random and RIQ dispatching for larger values of d because under these policies they compete with servers of lower capacities, with less probability of killing. Under AQL and QC, the competition will mostly be between servers of high capacity, which increases the wasted work in this cluster.

B. Time-progression Results

We show here a variety of graphs for a typical simulation run with 3,000 job completions. Results are shown for $d = 10$, $\lambda = 14$ requests/sec, AQL and QC policies, and are displayed for each 50 job completions. Because server selection under AQL and QC depends on the estimated response time, in addition to $P_{\text{complete},i}$ and $P_{\text{kill},i}$, we show how these values converge with time.

Figure 11 shows four different graphs numbered 1 through 4 from left to right and then top to bottom. We describe now each graph and the observations we derive from them.

- Fig. 11 (1): depicts the variation of the response time, normalized by the number of operations per job and averaged for all jobs completed at each cluster vs. the number of job completed. The results are shown for all the clusters and under AQL dispatching policy. It can be seen that the response time converges quickly.
- Fig. 11 (2): depicts similar results as those of Fig. 11 (1) but for the QC policy. The results take slightly longer to converge because the response time estimate depends on the dispatcher obtaining a stable estimate of $P_{\text{complete},i}$ and $P_{\text{kill},i}$ shown in the next two charts.
- Fig. 11 (3): depicts the variation of $P_{\text{complete},i}$ averaged for all jobs completed at each cluster vs. the number of job completed. It can be seen that $P_{\text{complete},i}$ for cluster 1 starts high until this cluster starts receiving jobs due to long queues in other clusters. However, most of the jobs are killed, and thus $P_{\text{complete},i}$ decreases. The other clusters decrease their completion probability before the completion of 50 jobs. Since these results are for $d = 10$, we expect significant job killing and wasted work.
- Fig. 11 (4): depicts the variation of $P_{\text{kill},i}$ averaged for all jobs completed at each cluster vs. the number of jobs completed. The changes in this chart happen in the opposite direction from the changes in Fig. 11 (3). Cluster 1 has a slow convergence mainly due to a low dispatch rate to servers in this cluster.

IX. Conclusions

This paper investigated the advantages of submitting multiple replicas of a job to the servers of a heterogeneous cluster in order to reduce the average response time. A dispatcher receives jobs but does not know which server would be able to process the job faster. So, submitting an arriving job to multiple servers increases the chance that the job will complete faster. Once a job completes, all its replicas, including the ones being processed, are removed from the other servers. If the number of replicas is too high, the number of jobs to kill will increase and the partially completed jobs delay other jobs waiting to use that server. Therefore, there is an optimal value of the number of replicas.

We considered here the possibility of jobs failing during processing and provided an analytic expression for the average time wasted due to failures. Extensive simulation experiments were carried out to compare four dispatching policies under various number of replicas. Two of the policies (Random and RIQ) require the dispatcher to know no or very little information about the server operation. For these policies, the average response time decreases with the number of replicas and increases again as the number of replicas increases further. The AQL and QC policies are more intrusive and require significantly more coordination between the dispatcher and each server. These policies do not show any benefit from using replication and are the best overall, as long as one is willing to tolerate the strong coupling between the dispatcher and the servers. The explanation for that is that AQL and QC operate on estimating the response time at each server. If the estimate were 100% precise, the best policy would be to send a single copy of the job to the server where that job would experience the smallest response time.

In the future, we intend to investigate policies that take cost into account. Cost-aware policies are especially important in IaaS clouds since the longer it takes to process a workload the higher the cost. In this paper, we considered that all jobs have the same distribution of number of operations. As future work, we intend to consider the multiclass case where jobs of different classes have different distributions of number of operations. Additionally, we intend to investigate the specific overhead of undoing actions taken by killed jobs. This is important for stateful jobs. Finally, we plan to investigate the error between the response time estimates computed by the AQL and QC policies and the actual response times.

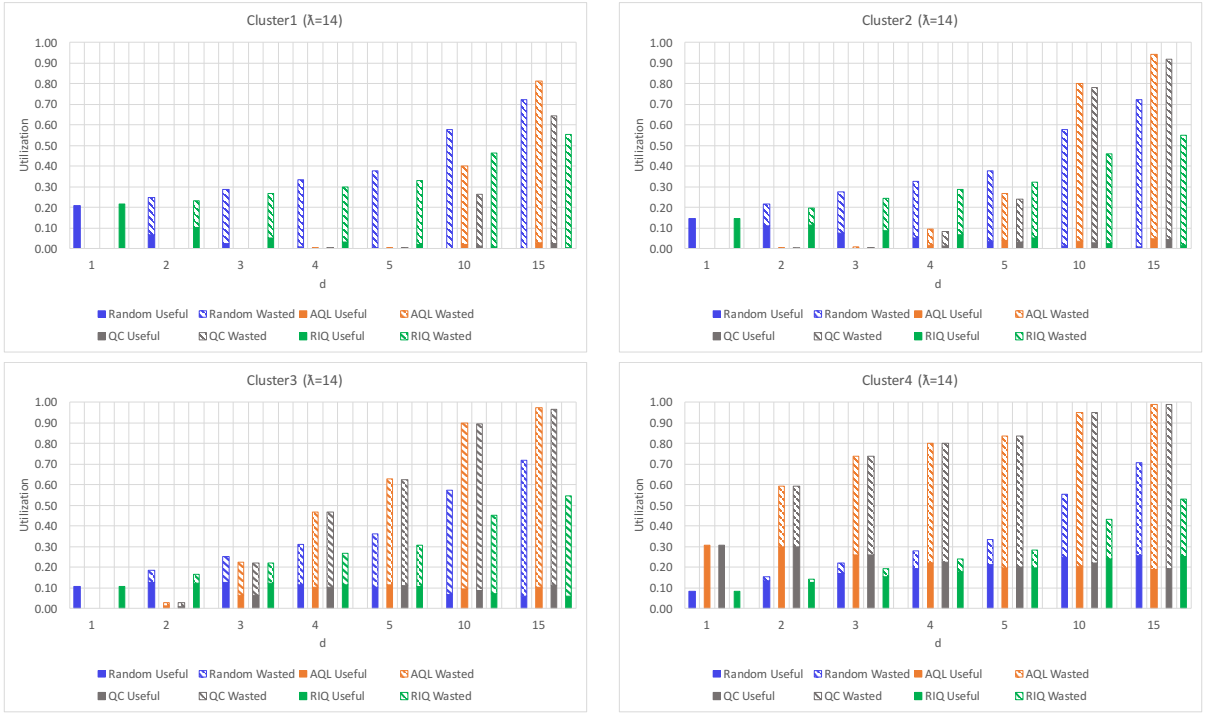

Fig. 10: Results for Average $U_{u,i}$ and $U_{w,i}$.

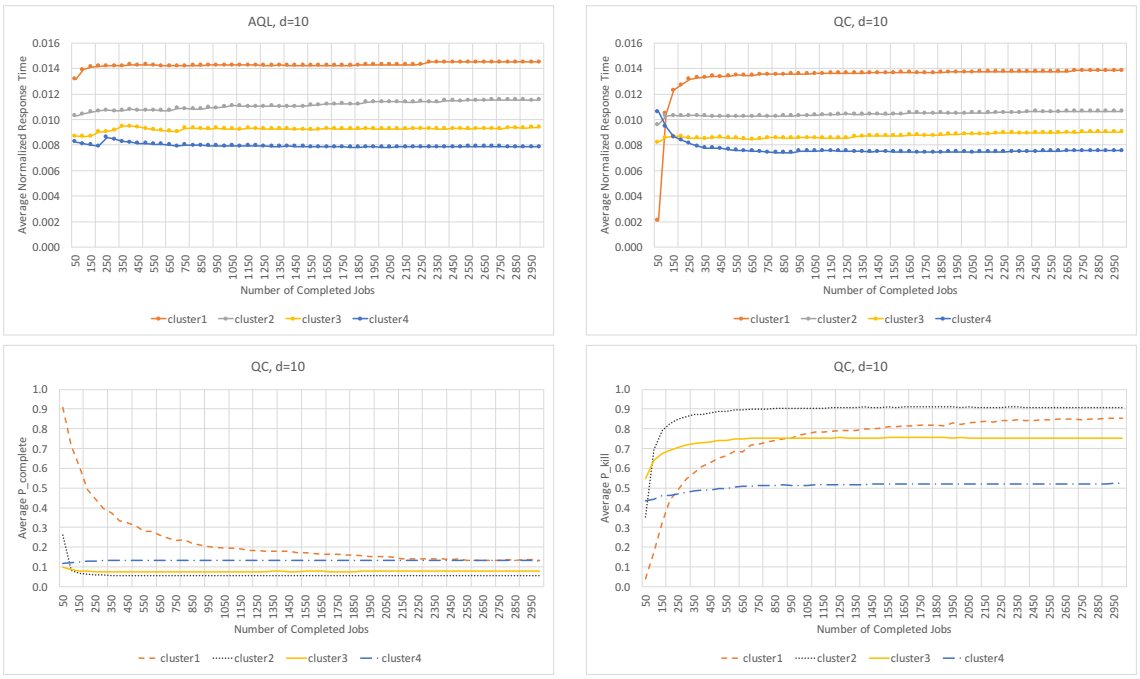

Fig. 11: Time-progression Results.

ACKNOWLEDGEMENTS

This work was partially supported by the AFOSR grant FA9550-16-1-0030.

REFERENCES

[1] L.A. Barroso, J. Clidaras, and U. Holzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd. edition, Morgan & Claypool Publishers series Synthesis Lectures on Computer Architecture, 2013, ISBN: 9781627050098.

[2] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, Communications of the ACM, 51 (1), pp. 107-113, 2008.

[3] Gardner, K., M. Harchol-Balter, and A. Scheller-Wolf, *A Better Model for Job Redundancy: Decoupling Server Slowdown and Job Size*, IEEE Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2016) . London, UK, September 2016.

[4] Gardner, K., S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytia, and A. Scheller-Wolf, *Reducing latency via redundant requests: Exact analysis*, Proc. ACM SIGMETRICS, June 2015.

[5] Harchol-Balter, Mor, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*, Cambridge University Press, 2013.

[6] Huang, L., S. Pawar, H. Zhang, and K. Ramchandran, *Codes can reduce queueing delay in data centers*, 2012 IEEE Intl. Symp. Information Theory (ISIT), pp. 27662770, 2012.

[7] Kleinrock, L., *Queuing Systems, Volume I: Theory*, John Wiley & Sons, 1975.

[8] Koole G. and R. Righter, *Resource allocation in grid computing*, Journal of Scheduling, 11:163173, 2009.

[9] Rayward-Smith, V.J., I.H. Osman, C.R. Reeves, and G.D. Smith, eds., *Modern Heuristic Search Methods,*, John Wiley, Hoboken, NJ, 1996.

[10] Zaharia, Matei, Andy Konwinski, Anthony D. Joseph, Randy H. Katz, and Ion Stoica, *Improving MapReduce performance in heterogeneous environments,*, In Osdi, vol. 8, no. 4, 2008.