Modular Data Centers: How to Design Them?

Kashi Venkatesh Vishwanath, Albert Greenberg, and Daniel A. Reed
Microsoft Research, Redmond, WA, USA
{kashi.vishwanath,albert,reed}@microsoft.com

ABSTRACT

There has been a recent interest in modularized shipping containers as the building block for data centers. However, there are no published results on the different design tradeoffs it offers. In this paper we investigate a model where such a container is never serviced during its deployment lifetime, say 3 years, for hardware faults. Instead, the hardware is over-provisioned in the beginning and failures are handled gracefully by software. The reasons vary from ease of accounting and management to increased design flexibility owing to its sealed and service-free nature.

We present a preliminary model for performance, reliability and cost for such service-less containerized solutions. There are a number of design choices/policies for over-provisioning the containers. For instance, as a function of dead servers and incoming workload we could decide which servers to selectively turn on/off while still maintaining a desired level of performance. While evaluating each such choice is challenging, we demonstrate that arriving at the best and worst-case design is tractable. We further demonstrate that projected lifetimes of these extreme cases are very close (within 10%) to each other. One way to interpret this reliability number is, the utility of keeping machines as cold spares within the container, in anticipation of server failures, is not too different than starting out with all machines active. So as we engineer the containers in sophisticated ways for cost and performance, we can arrive at the associated reliability estimates using a simpler more-tractable approximation. We demonstrate that these bounds are robust to general distributions for failure times of servers.

We hope that this paper stirs up a number of research investigations geared towards understanding these next generation data center building blocks. This involves both improving the models and corroborating them with field data.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability

General Terms

Design, Measurement, Performance, Reliability

LSAP'09, June 10, 2009, Munich, Germany.

Figure 1: Container units for data centers.

Keywords

Containers, Data Centers, Performance, Reliability

1. INTRODUCTION

In recent years, there has been a significant interest in modularized data centers by a number of cloud service providers and hardware vendors [13, 14, 18, 10, 1]. These proposed next-generation data center building blocks are enclosed in standard shipping containers (Figure 1). Containerization simplifies supply change management. Instead of packaging the servers and then having personnel install them at a remote location, the servers come already setup and wired. At the remote location, the containers become large pluggable components – upon standardized hook up to power, networking, and cooling infrastructure, service enabling can commence. With large services spanning thousands of machines, a container better approximates the right scale unit, rather than individual or racks of servers. Importantly, containers also hold the promise of lowering total cost of ownership, by lowering costs associated with the need for continuous component repair. For example, if incremental repair is not an objective, the container can be packed densely and run hotter.

Indeed, containerization allows us to think of designs that are *service-free*. That is, the container as a whole is not repaired on incremental component failure. Rather, it is kept in service, while enough components are in service that its service utility meets an engineered minimum level. When the container's utility degrades below that level, or it reaches the end of its intended life-cycle (e.g., three years), it can be returned and replaced with another one supporting newer technology. Service-free operation is particularly ap-

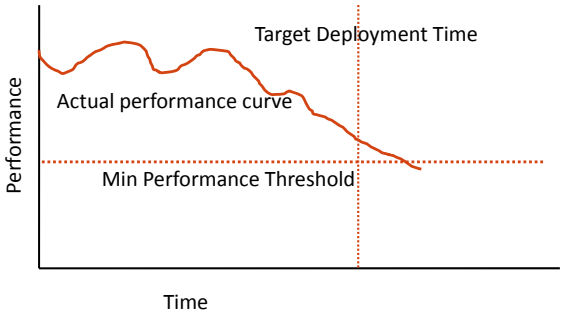

Figure 2: Performance vs. reliability design space.

pealing for distributed data centers in remote locations [14] that are not easily human accessible.

Overall, containerization shifts the problem of supply chain management and ongoing systems management upstream to the hardware vendor, who is then tasked with designing a solution that lowers total cost of ownership for the cloud service provider. Containerized solutions are rapidly becoming available from all large computer systems vendors. However, there has been no formulation of the tradeoffs in costs, reliability and performance.

A wide range of questions demand answers. Consider, for example, Figure 2, which shows the performance of a hypothetical container over time. Even if performance changes (and degrades) over time owing to hardware failures and design choices, the container is useful to the service provider as long as the performance is above the minimum threshold (shown by the dashed horizontal line) up to the target deployment time (shown by the dashed vertical line). How long can we assure that the container holds enough working components to meet the threshold? What is the minimum reliability specification for a server, which we intend to deploy in a group of 1,000 in a container, where we are willing to tolerate up to 100 failures through three years? How can we design servers to reach that level of reliability, i.e., how much would each cost? In estimating the total cost of ownership (TCO) we must include component cost as well as operational cost, including the power to drive the container. It is far from obvious how to choose components for these containers. If we pack containers with highly reliable servers, will the increase in reliability overshoot our target deployment lifetime, and so result in paying a premium without commensurate benefit? Viewed another way, if we increase cost through adding redundancy to each container, will we increase the total cost of the network of containers, squandering the potential savings for lowering costs per container and raising reliability through redundancy at container-level?

In this paper, we propose the first performance and reliability model of containerized data centers. Our mathematical models allow us to pose questions relating to estimated time needed until the container degrades to a level where repair is called for. The component failure model is fail-stop. (In practice, this is the last stage of the restart, reboot, re-image, and return life-cycle of the recovery oriented computing paradigm.) Though, we focus on service-free operation, the model applies equally well to deployments where containers are never serviced on site, and to deployments where on site service is an option, at a cost. While simple, the models are sufficiently large and complex to defy fast solution. Fortunately, however, the models admit corresponding lower and upper bounding models to be developed, which provide insight through simple formulas, relating critical parameters, which can be fit to measurement data. Our models aim to capture high order effects, and provide engineering insights, by sacrificing the plethora of detail that would eventually have to go into operational capacity planning and deployment. The results allow us to reason about the ability to meet performance targets over time.

Beyond the models presented, we work through initial analysis based on well known failure rates of components. This is a first step towards modeling data center containerization. In combination with lab-test and field-data on performance, reliability, cost and availability, we can evaluate how to engineer these units from individual components. For instance, taking into account workloads, should we use cheap HDDs (conventional, Hard Disk Drives) and mirror them for reliability, or should we buy expensive SSDs (flash-based, Solid State Drives)? We hope that as a result of this paper, a more comprehensive modeling effort may be carried out to address these sort of questions.

Once we have satisfactory answers to the problem of container design, the onus shifts to delivering infrastructure management and application software that can live up to the expectations of the model predictions on performance and reliability. Thus, we argue that the containerized model warrants further investigation from the point of view of super low cost service provisioning. Careful container design will not only save tremendous capital expense, but will also push us towards reducing software complexity and raising service reliability. It is therefore well worth the effort to investigate choices for reliable container design.

2. PROBLEM SETUP

We introduce the performance and reliability modeling problem using the notation shown in Table 1 below.

Table 1: Model variables for the container.

T	The target lifetime of deployment
B	Total budget for T years
C	Cost of a unit component(server)
p	Performance of each component
W	Power draw of each component
F	Annualized failure rate (AFR) of the component
P_{th}	Minimum desired performance of container
E	Electricity cost

Given these inputs we must decide on the number of servers to pack in the container and a server-usage policy that maximizes the container lifetime while maintaining the desired level of performance.

In order to make the problem tractable we make a few assumptions as a first approximation and later relax some of these assumptions. More importantly, this is to get an initial approximation of performance and reliability values. We assume that each of the components can fail independently and with an identical distribution. Furthermore, we assume that the failure distribution is exponential. We also assume that there are no correlated failures other than single points of failure. Section 5.1 examines the results when these assumption are relaxed including our analysis for general failure distributions.

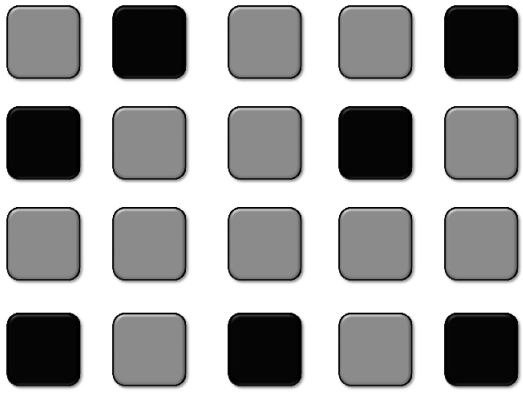

Figure 3: Lower bound for MTTF ($MTTF_{lb}$). All servers start out as active.

3. RELIABILITY, PERFORMANCE AND TCO

Recall that we have a service-less model for the containers where we provision the container with spare servers. There are a number of different schemes to do so. Say we start with N servers. Let us define the unit to be functional as long as at least k servers are functional at any given time. Thus, using our notation from Table 1, $k = \frac{P_{th}}{p}$.
We investigate how to provision/design a container such that it still yields a performance of P_{th} at the end of T years.

3.1 Reliability

To understand reliability, we would like to calculate the mean time to failure (MTTF) for various design choices of these containerized units. This would require us to understand a variety of different schemes to over-provision for future failures. There are a variety of concerns on how to detect failures, and transfer load seamlessly, or bring cold spares online. However, that is not the concern of this paper. The aim here is to understanding the guarantees such a scheme could ever provide, assuming that robust software can be written for it. In order to distill down from the myriad choices of provisioning the container we examine two ends of the spectrum of design choices to evaluate the cost/reliability benefits.

At one extreme we consider the simplest possible scheme that results in the least utilization of servers. Thus, we start with all N servers as active. Next, load is spread evenly so that all machines are utilized. The total performance that this container can deliver is $p \times N$, which is greater than P_{th}. As and when machines start failing the load is distributed evenly onto the remainder of the machines that are still active. For instance, if $N = 20$, and $k = 12$, Figure 3 shows that 7 of the 20 servers that we started out with, have died over time (black). The container is still functional in this state as it has 1 more than the minimum 12 servers required to meet the target performance level. Eventually, when only 11 servers remain active we declare the unit as dead. Such a design choice leads to a wastage of resources in the early part of the deployment lifespan as the total performance the container could yield is more than the requirement. Furthermore, keeping the servers active would just make them susceptible to failure due to mechanical moving parts. The lifetime of such a container will be the lowest possible among all possible design choices. Thus, the mean-time-to-failure (MTTF) for this scheme, is the lower bound ($MTTF_{lb}$) for all possible MTTF values.

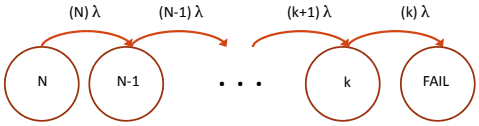

Figure 4: Markov chain for minimizing MTTF.

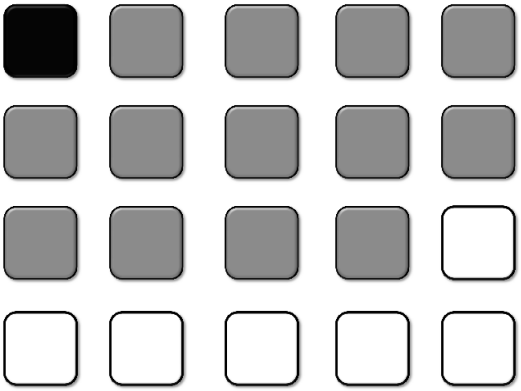

Figure 5: Upper bound for MTTF ($MTTF_{ub}$). Only $k+1$ servers are kept active. Upon failure of a server, a cold one is brought online in its place.

Given our assumption of exponential failure times (relaxed in Section 5.1) for all components, we can model this as a Markov chain as shown in Figure 4. The state is described by the number of functional servers in each state. The failure of each server is assumed to be independent with an exponential distribution with mean $1/\lambda$. Thus the time to failure of the first server is distributed exponentially with mean $1/N\lambda$. The transition represents the failure of a server, thus, changing the number of active servers in the container. Finally, when the server is in state k, the failure of a single server renders the container unusable. Thus $MTTF_{lb}$ is simply given by [19],

$$\frac{1}{N\lambda} + \frac{1}{(N-1)\lambda} + \cdots + \frac{1}{k\lambda},$$

$$MTTF_{lb} = \frac{1}{\lambda} \sum_{i=k}^{i=N} \frac{1}{i} \quad (1)$$

At the other extreme, in order to maximize the MTTF we consider the following scheme. We start with only $k+1$ servers as active and all others (i.e., $N-k-1$) are kept as cold spares. As soon as a server fails we bring a cold server online and substitute it for the failed server, thus restoring the number of active servers back to $k+1$. We are assuming that as soon as a server fails, there is 100% fault detection and coverage and we can instantaneously bring a cold server online to replace it. Furthermore, we assume that cold spares never fail before coming online. Thus, this is the most optimistic design we can ever hope to achieve and the corresponding MTTF will give us an upper bound ($MTTF_{ub}$) on all possible MTTF values.

Figure 5 shows the state of the system when one server has failed (black). Thus, 13 of the first 14 servers are active (gray) and the remaining 6 are cold-spares (white). At all times we keep only 13

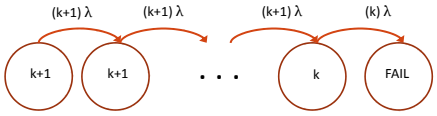

Figure 6: Markov chain for maximizing MTTF.

$(i.e.,\ k+1)$ servers active with the remainder of the not-yet-failed servers kept as cold spares. This scheme is extremely conservative in the number of servers that are brought online.

Given our assumption of exponential failure times, this too can be expressed as a Markov chain as shown in Figure 6. The state describes the number of servers that are active. The transitions represent the failure of a server and a cold spare taking its place seamlessly. Thus, $MTTF_{ub}$ is,

$$\frac{1}{(k+1)\lambda} + \frac{1}{(k+1)\lambda} + \cdots + \frac{1}{(k+1)\lambda} + \frac{1}{k\lambda},$$

$$MTTF_{ub} = \frac{N-k}{(k+1)\lambda} + \frac{1}{k\lambda} \tag{2}$$

Thus, we have derived lower ($MTTF_{lb}$) and upper ($MTTF_{ub}$) bounds for the value of MTTF for N servers with at least k functional units. For instance, say k is 85% of N and $F = 5\%$. Figure 7 shows that as the number of servers, N grows, the MTTF of the container begins to drop but the curve eventually flattens out. Thus, as long as we are in the flat part of the curve, increasing the size of the container does not adversely effect the MTTF as long as we are willing to tolerate the same percentage of failures (15% in this case).

The next question we ask is the following. What is the gap between the lower and the upper bounds. If the gap is tight then we do not need to investigate the wide spectrum of design choices in detail before arriving at reliability guarantees. On the other hand, if the gap is wide then further fine-tuning of the bounds is warranted. The results are shown in Figure 7. When the number of components is small, the gap is quite significant (5.5 years for upper bound vs. 4.5 years for lower bound when number of components is 10). However, as the number of components increases, this gap narrows and almost becomes a constant once the number of components grow beyond 200 (3.5 years for upper bound vs. 3.25 years for lower bound). We did a similar analysis with a variety of different scenarios and for reasonable annualized failure rate (AFR) of below 10%, the gap between the lower and upper bound was within 10% of each other. The main intuition behind this is as follows. A large portion of servers are always ON. Thus, the cold-spare scheme can only prolong the lifetime of a small percentage of servers thereby not impacting overall container lifetime significantly. Thus, we can use either of the lower/upper-bound values as an estimate of the container's MTTF.

3.2 Performance

To understand performance of a system where individual components could fail, an integrated performance and reliability model, also know as a performability [12] model is preferred. We use a popular variant of this known as Markov Reward Model (MRM), that builds on the Markov chain by associating rewards to each state. The reward earned at a particular state is proportional to the amount of time spent in that state. Overall performability is the total reward earned during the lifetime.

To get the performability for the lower bound variant let us associate the reward with each state in Figure 4 as the performance of the container when in that state. We can assume that the performance is proportional to the number of active servers at that instant.

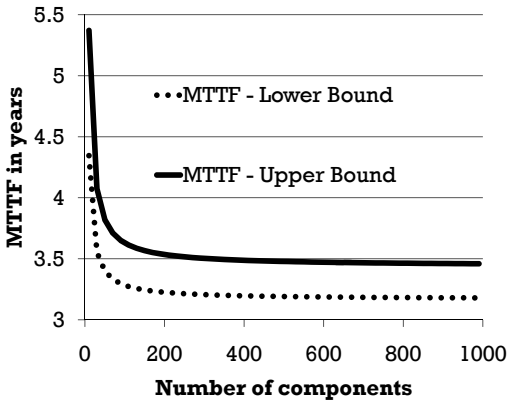

Figure 7: Gap in lower and upper bounds of MTTF.

Thus, the reward in state i is proportional to i, say $a \times i$, where a is a constant. The overall reward is given by, $a \times N \times \frac{1}{N\lambda} + a \times (N-1) \times \frac{1}{(N-1)\lambda} + \cdots + a \times k \times \frac{1}{k\lambda}$

$$= a \times \frac{1}{\lambda} + a \times \frac{1}{\lambda} + \cdots + a \times \frac{1}{\lambda}$$

$$= a \times (N-k+1) \times \frac{1}{\lambda} \tag{3}$$

Similarly, let us calculate the performability of the upper bound variant by again building a MRM and associating a reward $a \times i$ to state i, this time in Figure 6. The overall reward is given by,

$$a \times (k+1) \times \frac{1}{(k+1)\lambda} + \cdots + a \times (k+1) \times \frac{1}{(k+1)\lambda} + a \times k \times \frac{1}{k\lambda}$$

$$= a \times \frac{1}{\lambda} + \cdots + a \times \frac{1}{\lambda} + a \times \frac{1}{\lambda}$$

$$= a \times (N-k+1) \times \frac{1}{\lambda} \tag{4}$$

Thus, the total performance of the two extremes (lower and upper bound MTTF) are the same over their corresponding lifetimes. Intuitively, in the lower bound case the performance is very high when the container is just deployed ($N \times a$) and it keeps falling rapidly with time. On the other hand, the performance is constant in the upper bound case ($\{k+1\} \times a$) but is delivered over a longer lifetime.

3.3 Power Cost

In order to calculate the power draw for the lower bound and the upper bound we again build a MRM. The reward at each state is the amount of energy drawn. This is proportional to the number of servers active in a given state. Thus, the results will be identical to those for performance, albeit with a different constant. Thus, similar to performance, the total energy spent in running the machines is the same over the lifetime of the two design choices.

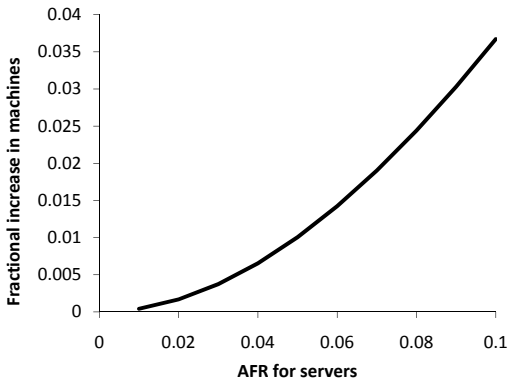

Figure 8: Fractional increase in machines from upper bound to lower bound.

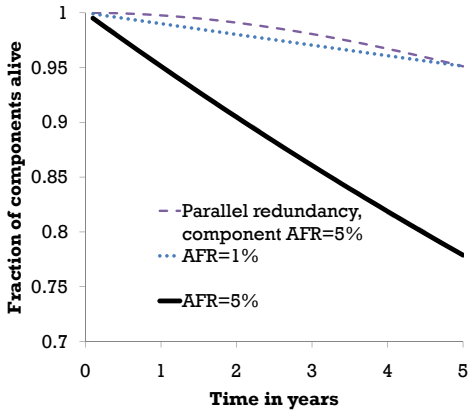

Figure 9: Reliability of RAID1 with 5% AFR HDDs is equivalent to a single HDD with AFR 1%.

3.4 Component Cost

Figure 7 represents one way of comparing the schemes. In reality however, we would know the time of deployment, i.e., T in our model. Amortizing the cost of replacing hardware and upgrading to the latest available technology mandates that T be in the 3-5 year range. Let us say that $T = 3$ years. Given the failure rate F, and threshold k we can calculate how many machines to start with i.e., N, for both the lower bound as well as the upper bound models. Since the two numbers will be different, we can compare the percentage difference to get an estimate of how "expensive" one option is vs. the other in terms of component cost. Figure 8 shows the fractional increase in machines in moving from the upper bound to the lower bound model against different failure rate for servers. For instance, for failure rate of around 5%, we would need just 1% (i.e., 0.01 fraction) more servers in the lower bound model than the upper bound model to achieve the same MTTF. Even with failure rate of 10%, the increase in number of servers is around 4%.

Based on the above results we conclude that for reasonable values of failure rate (<10% AFR), using the lower bound variant is a good approximation for the cost, reliability, and performance of any arbitrary scheme. A majority of the remainder of this paper will show further analysis with this assumption.

4. DESIGNING A CONTAINER

Following the methodology outlined thus far, we can get the total cost of ownership (TCO) as the component cost + operations cost (i.e., the power cost). We next demonstrate how to use TCO to compare different design choices in building a container.

Consider a typical server with hard-disk reliability governed by an AFR of 5% and for all other components an aggregate of 5% AFR. The reliability of the server is a product of the reliability of the disk and other components. Thus, it is exponentially distributed with mean,

$$\frac{1}{\lambda_{disk} + \lambda_{other}}$$

where λ_{disk} and λ_{other} can be calculated from the corresponding failure rates (AFRs) using the following equation,

$$AFR = 1 - e^{\lambda \times 1}$$

Say, the server costs $\$1,000$ and draws 100W of power. Also, assume that the server is put to I/O intensive jobs and for a random workload gives us a performance of 500 IOPS. Let us say that we wanted to build a container of these server components that lasts 3 years and has a $P_{th} = 500,000$ IOPS. Since the performance of each server is 500 IOPS, this implies that k is $1,000$. Based on the AFR, we can use the lower bound variant to calculate N, which in this case equals 1370 servers (Equation 1). Thus the total component cost is $1370 \times 1,000$ =$\$1,370,000$ and the power cost (based on the MRM described in Section 3.2) is $\$185,076.54$ assuming $\$0.06$ per KW-h. Thus, the cost is dominated by component cost. Likewise, if we had other choices of servers, then based on their reliability, cost, performance and power ratings we could compare them.

One option to avoid buying a lot of servers is to increase the reliability of individual servers by making the storage subsystem more reliable. One way of doing that might be to consider a RAID1 system i.e., by mirroring the hard disk. This would mean more cost to buy extra hard disks for each server, and more power draw, but an increased reliability due to redundancy (one hard-disk can take over when the other one fails). Note, that the performance of the server does not change at all. If we assume that the disk reliability is exponentially distributed with parameter λ_{disk} then the reliability of the parallel redundant RAID1 subsystem can be expressed as thus,

$$1 - (1 - e^{-\lambda_{disk} t})^2$$

This is no longer exponential, hence not as easily tractable. However, for the 5% AFR of a single disk, as shown in Figure 9 the reliability curve of this parallel redundant system can be bounded below by an exponential curve corresponding to an AFR of 1%. Since we are using a lower bound on MTTF we further relax the bound by using the exponential distribution corresponding to AFR of 1% as an approximation to the RAID1 reliability. Now we can get the server reliability as a series combination of the RAID1 subsystem and the rest of the component. This corresponds to a $\sim 6\%$ AFR. In adding the extra disk and the RAID controller, say the cost of the server goes up by $\$150$, and the power draw increases by 20W. We repeat our analysis and this time it turns out that we need to start with only $1,203$ servers. Our component cost is $\$1,383,450$ and the power cost is $\$207,945$. Thus, both the costs have actually gone up slightly but we have fewer servers to fit in the container now.

Finally, we consider a third alternative, this time replacing disks with SSDs. We can assume a 20W savings in power. Also we can

Design Choice	Server Cost (USD)	AFR	Power draw (W)	Performance (IOPS)	k (at end of 3 years)	N (total components)
SSD	2500	0.06	80	1,000	500	601
RAID 1	1150	0.06	120	500	1,000	1,203
HDD	1,000	0.1	100	500	1,000	1,370

Table 2: Cost, reliability, and performance for design alternatives.

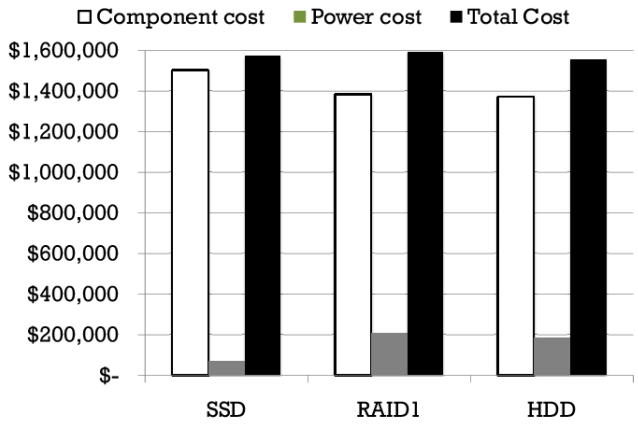

Figure 10: Bar chart showing TCO as well as distribution of component and power cost for the various choices.

assume that the reliability is no worse than 1% In other words, the reliability is the same as a RAID1 provisioned server. However, let us say that it costs an extra $\$1,500$ to buy an SSD (hiking the server cost to $\$2,500$) instead of a HDD but you get twice the performance[1]. Thus, k is only 500 now and the total number of components to start with is 601. The component cost is $\$1,502,500$ and the power cost is only $\$69,315$.

The parameters for these three options are summarized in Table 2. The last column also shows the total number of servers of each type that we need to put in the container to meet the target MTTF and performance requirement. The corresponding component and power costs are shown in Figure 10. Thus, the total cost of ownership (TCO) for the three options are very close to each other. One way to interpret the result is that the current choice of values represents the break even point for the three options. For instance, if SSDs start costing a little less, or give more than double the performance of HDDs for random I/O workloads then it would make sense to migrate to SSDs. The other way to look at this is that it makes sense to migrate given that it will reduce the number of servers, make individual servers more reliable, (thereby placing less stress on the software complexity) and finally be a greener solution (by reducing the power consumption)!

5. RELAXING THE ASSUMPTIONS

The models presented thus far put us into position to analyze different design choices for container based data centers. However, we made a number of simplifying assumptions. While those provide a first approximation, further refinement is clearly required. In this section we relax two of the assumptions, regarding exponential failure rates and correlated failures.

[1]Not true for sequential workloads. We do not consider disk capacity either. Extending our analysis to include these are avenues of future work.

5.1 Failure Distribution

Thus far, we have assumed server failure rates are exponential. However, a simple methodology, borrowed from the asymptotic analysis of order statistics, provides sharp estimates, which work for general failure distributions.

Let T model the time to fail a given component. Consider the cumulative distribution function for failures, $F(x)$, and the complementary cumulative distribution function, $G(x) = Pr(T > x)$, i.e., the probability that the time to fail is greater than x. Assume we have N components, with independent, identically distributed times to failure T_i, each modeled by G. Define $x_i = 1$ if $T_i > m$, and 0 otherwise.
Thus, the expected number of failed components after time m,
$E(\#T_i > m) = E(\Sigma_i x_i)$
$= n \times Pr(T > m)$
$= n \times G(m)$.

For, our lower bound we seek the time to hit the condition where k servers remain alive. We estimate that time by solving
$n \times G(m) = k$
where m is the MTTF. For instance, if T is exponential, as we had assumed so far, then
$G(x) = exp(-\lambda x)$, thus $m = -log(k/n)/\lambda$.

Recent studies have found that disk failure rates are better described with families of distributions, whose parameters can be set to behave much differently than the exponential. In particular, the Weibull distribution [16] provides good fits to data. It is worthwhile, moreover, to consider disk failures, because these tend to dominate all other hard systems failures [15, 11]. Approximating the failure rate of servers as a Weibull distribution, we obtain:

$F(x) = 1 - e^{-\lambda t^\alpha}$.
$=> G(x) = e^{-\lambda t^\alpha}$.
$=> k = n \times e^{-\lambda t^\alpha}$.
$=> t = -log(k/n)/\lambda^{1/\alpha}$.

Taking this estimate of the MTTF, let's examine the impact. Specifically, set values of $\lambda = 0.049$, and $\alpha = 0.71$, as reported in [16]. The derived value of MTTF will be lower than the result obtained using the exponential distribution. A similar generalization of the analysis might be carried out for the upper bounding model. However, we observe that the exponential provides a larger upper bound than would be obtained with an appropriately parameterized Weibull. That is, the MTTF for the upper bound is higher when we assume an exponential distribution, as opposed to the appropriate Weibull. Thus, the estimate of the gap between lower and upper bounds is conservative if we use exponential assumptions for the upper bound. Similar to our analysis in Section 3.4 (Figure 8) we analyze the number of extra nodes for the lower bound against the upper bound. Figure 11 shows fractional difference in the number of nodes as a function of k. If we assume exponential failure for both the upper and lower bound then the gap is just around 1% for all values of k. However, if we relax the lower bound to Weibull distribution but retain the upper bound as exponential, even then

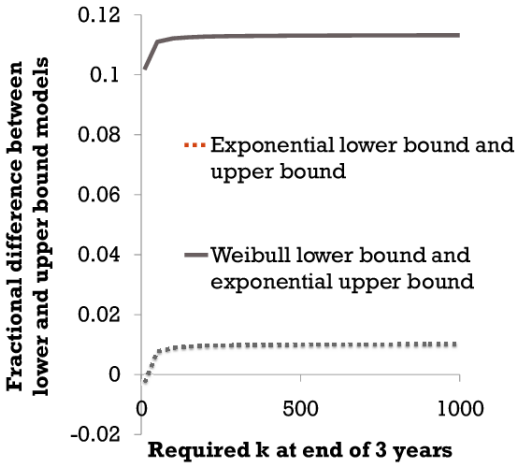

Figure 11: Comparing difference in using Weibull vs. exponential as lower bound.

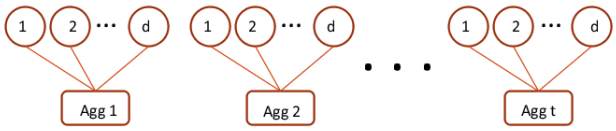

Figure 12: Points of failure in the design. In reality t varies from 1 to N.

the gap is still only 10%. While the relative change to the estimate is large, the takeaway – the gap itself is small in absolute terms – remains in force.

5.2 Correlated Failures

We use the following framework to help understand correlated failures. As shown in Figure 12, suppose that, for every subset of d servers there is a single point of failure, say a network switch, whose failure is equivalent to the failure of d servers. As a first approximation to this generic framework we analyze the following variation of our model. Suppose that there exists a single point of failure for the entire container. Let us retain exponential distributions for component times to fail, for simplicity. We model this via the Markov Chain shown in Figure 13, which agrees with that of Figure 4 except that from each state there is an extra transition to the FAIL state with failure rate λSPF (Single Point of Failure).

We can easily calculate the MTTF by solving balanced transition equations for this Chain [19]. The MTTF is naturally lower then that of Figure 4. Putting it another way, the lifetime of the container is dictated by reaching a threshold number of failed servers, or by the SPF failing, whichever happens first. However, with a careful design the reduction in MTTF can be largely controlled. For instance, with a 5% AFR for the servers as well as the single point of failure, the MTTF decreases by 5 months from a 3 year target deployment lifetime. Adapting the analysis to use use redundancy for the single point of failure reduces aggregate AFR from 5% to 1%, and reduces MTTF by just 1 month.

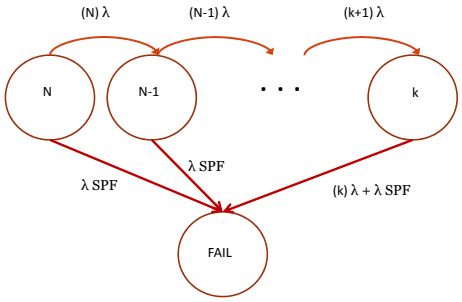

Figure 13: Markov chain when a single point of failure can bring the entire container down.

More fundamentally, correlation in failures can arise from shared dependencies stemming from manufacturing batch, environmental, power, and networking dimensions. These can be mitigated at container-level: We can reliably create systems of multiple containers. These can also be mitigated through use of diversity across these dimensions – e.g., use of components from different manufacturing batches.

6. DISCUSSION

The models considered here provide context for analyzing and designing container-based data center building blocks. They highlight the importance of considering the interplay of reliability, performance and cost as elements of total cost of ownership (TCO). More generally, the types and characteristics of workloads, service level agreements, time to deployment and operating cost must be incorporated into richer models. The goal of this paper is to illuminate the issues and suggest fruitful topics for further exploration within a framework for multivariate optimization.

To simplify the models and bound the range of behaviors, we have made several simplifying assumptions about performance in response to offered load and the probability of component failure in response to that load. Clearly, additional data are needed on container failure modes and rates, drawn from field experience. In particular, this failure data would enable construction of hierarchical reliability models for nodes, based on processor, memory, storage, network, power supply and cooling systems failures. With such hierarchical models, one can begin construction of improved analytic models and parametric simulation studies [6, 17] that increase confidence and reduce the errors in our bounding models.

As noted earlier, modeling reliability is but one aspect of integrated assessment. Time-varying resource demands and the resilience of application workloads to component failure also shape the feasibility of certain design alternatives. One intriguing aspect is considering periodic maintenance and "crop rotation" of containers based on integrated performability models and residual value relative to newer and presumably more efficient infrastructure.

The industry's longstanding hardware design point has been creating most reliable hardware possible within a given price envelope. One intriguing possibility is to consider the inverse, reducing reliability if one can dramatically decrease capital costs. However, this increases the importance of system and application software resilience to failures.

7. RELATED WORK

Recently, several vendors have developed data center building blocks based on industry-standard shipping containers [9, 1, 10, 13, 14, 18, 20]. This approach is motivated by the rapid increase in data center size, with cloud data centers now containing hundreds of thousands of servers. Larger building blocks simplify installation and configuration, reducing the time from purchase to operation. In addition, defining standard interfaces for networking, electrical power and cooling allow hardware vendors to innovate within the container while enabling interoperability across vendors and container generations. Finally, containers allow cloud service providers to build data centers and edge networks at a variety of scales, from a single container to hub-and-spoke mega-data centers [5, 8, 2].

Given the recent appearance of the container model, there has been little formal, parametric assessment of the balance of cost, reliability and performance at scale. At massive scale, assessing container reliability and the desirability and frequency of field component replacement while maintaining service level agreements (SLAs) are critically important issues. Quite clearly, at large scale, component failures are common [7] and designs and operational practice must reflect this reality.

Recovery oriented computing [4, 3] is one such instance, albeit at much smaller scale. Our work considers true "lights out" container farm design where systems must operate for extended periods without human intervention or field service. As such, it draws on analysis of component failures (e.g., DRAM, power supply, fans), cooling elements and storage systems [15, 16, 11].

8. CONCLUSION

Modularized data centers are gaining popularity, and essentially all major server hardware vendors have a container offering. To our knowledge, this paper is the first to present corresponding models of performance and reliability. Containerization leads quickly to the question of how to totally disrupt the model of supply chain and life cycle management of IT infrastructure, to a new model where the container is service-free. When performance degrades to an engineered threshold, or when the technology becomes outdated (whichever comes first), the container is returned or recycled. Our paper provides models that allow engineers to gauge how to build for such an objective (and related less radical objectives) economically. We showed that for typical component failure rates, the behaviors of the best and worst case strategies for use of preprovisioned extra components are remarkably close to each other. We further showed how to estimate total cost of operation based on both component and power costs. A preliminary example application of the results showed how to meet the same reliability objective at lower cost using multiple Solid State Drives, as opposed to multiple (RAID1) Hard Disk Drives, provided the workload can be accommodated adequately on both. We hope that, while simple, the models and the bounds derived are useful and robust for providing guidance on the tradeoffs between cost and reliability in containerized components. To go deeper, we see a plethora of opportunities to bring more detailed analysis and measurement into play.

9. REFERENCES

[1] ACTIVE POWER, HP TEAM ON POWERED CONTAINERS. http://www.datacenterknowledge.com/archives/2009/02/11/active-power-hp-team-on-powered-containers/.

[2] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the Clouds: A Berkeley View of Cloud Computing [White Paper] http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

[3] BROWN, A., AND PATTERSON, D. A. Embracing Failure: A Case for Recovery-Oriented Computing (ROC). In *High Performance Transaction Processing Symposium* (2001).

[4] BROWN, A., AND PATTERSON, D. A. Undo for Operators: Building an Undoable E-mail Store. In *2003 USENIX Annual Technical Conference* (2003).

[5] CHURCH, K., HAMILTON, J., AND GREENBERG, A. On Delivering Embarassingly Distributed Cloud Services. In *Hotnets VII* (2008).

[6] CSIM 19, MESQUITE SOFTWARE. http://www.mesquite.com/.

[7] FAILURE RATES IN GOOGLE DATA CENTERS. http://www.datacenterknowledge.com/archives/2008/05/30/failure-rates-in-google-data-centers/.

[8] GREENBERG, A., HAMILTON, J., MALTZ, D. A., AND PATEL, P. The Cost of a Cloud: Research Problems in Data Center Networks. In *SIGCOMM CCR* (2009).

[9] HAMILTON, J. An Architecture for Modular Data Centers. In *CIDR 2007*.

[10] HP POD, PERFORMANCE-OPTIMIZED DATA CENTER. http://h30424.www3.hp.com/pod/.

[11] JIANG, W., HU, C., ZHOU, Y., AND KANEVSKY, A. Are Disks the Dominant Contributor for Storage Failures? A Comprehensive Study of Storage Subsystem Failure Characteristics. In *File and Storage Technologies* (2008).

[12] MEYER, J. F. On Evaulating the Performability of Degradable Computing Systems. In *IEEE Transactions on Computers* (1980).

[13] MICROSOFT GOES ALL-IN ON CONTAINER DATA CENTERS. http://www.datacenterknowledge.com/archives/2008/12/02/microsoft-goes-all-in-on-container-data-centers/.

[14] MODULAR DATA CENTER (GOOGLE). http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=7,278,273.PN.&OS=PN/7,278,273&RS=PN/7,278,273.

[15] PINHEIRO, E., WEBER, W.-D., AND BARROSO, L. A. Failure Trends in a Large Disk Drive Population. In *File and Storage Technologies* (2007).

[16] SCHROEDER, B., AND GIBSON, G. A. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? In *File and Storage Technologies* (2007).

[17] SHARPE SOFTWARE. http://shannon.ee.duke.edu/tools/sharpe.

[18] SUN MODULAR DATACENTER. http://www.sun.com/products/sunmd/s20/.

[19] TRIVEDI, K. S. *Probability & Statistics with Reliability, Queueing, and Computer Science Applications*. Prentice Hall, 1982.

[20] UC SAN DIEGO'S GREENLIGHT PROJECT TO IMPROVE ENERGY EFFICIENCY OF COMPUTING. http://ucsdnews.ucsd.edu/newsrel/science/07-08GreenLightProj.asp.