Modeling Service Availability in Web Clusters Architectures

Magnos Martinello[1]*, Mohamed Kaâniche[2] and Karama Kanoun[2]

[1] Fundação Instituto Capixaba de Pesquisas em Contabilidade, Economia e Finanças
FUCAPE
Av. Fernando Ferrari, 1358, Goiabeiras, Vitória - ES

[2]LAAS-CNRS
7, Av. du Colonel Roche 31077 Toulouse - France

magnos@fucape.br, (kaaniche,kanoun)@laas.fr

Abstract. *Internet is often used for transaction based applications such as online banking, stock trading, among many others where the service outages are unacceptable. It is important for designers of such applications to analyze how hardware, software and performance related failures affect the quality of service delivered to the users. This paper presents analytical models for evaluating the service availability of web cluster architectures. A composite performance and availability modeling approach is defined considering various causes of service unavailability. In particular, web cluster systems are modeled taking into account: two recovery strategies (client transparent and non client-transparent). Sensitivity analysis results are presented to show their impact on the web service availability. The obtained results are aimed at providing useful guidelines to web designers.*

Keywords: *Modeling, availability, fault tolerance, web clusters.*

1. Introduction

Web systems with multiple nodes are leading architectures for building popular web sites that have to guarantee scalable, highly available and reliable services. Web clusters consisting of multiple nodes have proved to be a promising and cost-effective approach to support such services [16, 17, 18, 4]. Many giant service providers run on a cluster of nodes (e.g. Yahoo, eBay, Alta Vista, Netscape) supporting millions of requests per day.

One of the main problems faced by the web systems designers is to find an adequate sizing of the architecture to ensure high availability and performance for the delivered services. Evaluation techniques have shown to be well suited to address this problem and to find the right tradeoffs for achieving availability while providing acceptable levels of performance.

A significant body of work has focused on various aspects of web cluster performance evaluation. Particular attention has been devoted to the performance analysis considering different algorithms for load balancing among the servers [18]. Although

*M. Martinello had a fellowship from CAPES-Brazil.

many efforts have been dedicated to analyze the availability of web hosts using measurement based techniques [17, 9], less emphasis was devoted to the modeling of the web service availability taking into account the impact of server node failures and performance degradations [16, 12].

In this work, models are developed considering explicitly the cluster characteristics especially related to the recovery strategies. We concentrate our attention on two recovery strategies, referred to as non client transparent and client transparent. In the first strategy, all the submitted requests are lost as long as a node failure has not been detected. In the second strategy, the submitted requests are smoothly migrated to the remaining nodes in a user transparent way. Closed-form equations are obtained for web service availability in both recovery techniques, for which sensitivity analyzes are conducted with respect to cluster characteristics, taking into account explicitly:

1. the number of nodes in the cluster, the recovery strategy after a node failure, as well as the reliability of cluster nodes;
2. various causes of request loss due to buffer overflow, or node failures as well as during recovery time.

The rest of this paper is organized as follows. Section 2. presents an overview of web cluster architectures focusing on fault tolerance strategies. Section 3. introduces the modeling assumptions. Section 4. describes the modeling approach providing closed-form equations for web service availability. Section 5. presents sensitivity analysis results and section 6. concludes the paper.

2. Fault tolerance strategies in web clusters

A cluster-based web system (briefly, *web cluster*) refers to a collection of machines that are housed together in a single location, interconnected through a high-speed network, and present a single system image to the outside [18].

A typical web cluster consists of a front-end dispatcher (sometimes called load balancer) and many back-end servers. Over the past few years, a number of web cluster architectures have been proposed. These architectures implement a large variety of strategies for routing new requests (load balancing) and for fault tolerance.

2.1. Non Client Transparent (NCT) recovery strategy

Non client transparent (NCT) recovery strategy corresponds essentially to traditional web cluster solutions. These solutions do not provide transparent handling of requests at the time of node failure.

Round robin DNS [5] and DNS aliasing are examples of architectures in which the new requests are routed to available servers. If server replicas are running on multiple hosts, these schemes can be used to provide fault tolerance for web service by changing the host name to IP address mapping depending on the state of the system. When a server failure is detected, the host name is no longer mapped to the IP address of the failed server host. This scheme requires that clients re-issue the request if they do not receive a reply. In practice, clients may continue to see the old mapping due to DNS caching at the clients and DNS servers. This reduces the effectiveness of this scheme since after a failure, the client may need to re-issue the request several times before it is routed to a new server.

Centralized schemes, such as Magic Router [6] and Cisco Local Router [8], require requests to travel through a central router where they are routed to the desired server. Typically, the router detects server failures and does not route packets to servers that have failed. The central router is a single point of failure and a performance bottleneck since all packets must travel through it. Distributed Packet Rewriting [2] avoids single entry point by allowing the servers to send messages directly to clients and by implementing some of the router logic in the servers, so that they can forward the requests to different servers. However, these architectures are only capable to provide high availability via redundancy. Actually, a failed component can be replaced with the available redundant component. None of these schemes support recovering requests that were being processed when the failure occurred, in other words all ongoing requests on the failed node will be lost (i.e., these requests must be resent by the users).

2.2. Client Transparent (CT) recovery strategy

Client transparent (CT) recovery strategy supports requests migration. It enables web requests to be smoothly migrated and recovered on other working node(s) in the presence of server failure, in a user transparent way. Recently, there have been cluster implementations providing client transparent recovery strategies, e.g. [1, 10]. These architectures support requests migration for static and dynamic objects.

In [10], if a node fails while processing a request related to static objects, then the dispatcher will select a new server node with an idle pre-forked connection connected with the target server re-binding the client-side connection to the new selected server connection. After the new connection binding is determined, the dispatcher issues a ”range request” on the new server connection. The ”range request” is defined in the http 1.1 protocol allowing a client to request portions of a resource. Using this property, we can enable a request to continue downloading a file from another node. The web requests for dynamic content, for which responses are created on demand (cgi, scripts, asp) are mostly based on client-provided arguments. The size and content of such response is variable. A request related to a dynamic object is not ”idempotent”, i.e. sometimes the result of two successive requests may be different due for example to updates of the database. Consequently, they solve this problem using ”store and forward” of the response of a dynamic request. In other words, the dispatcher will not relay the response to the client until it receives the complete result.

On the other hand, the basic idea in the design of [1] is to use the error handling mechanisms of TCP to ensure that the backup has a copy of each request before it is available to the primary. Once a reply is generated by the primary, a complete copy is sent to the backup before any reply has been sent to the client. If the primary fails before starting to transmit the reply, the backup can transmit its copy. If it fails while sending the reply, the error handling mechanism of TCP is used.

3. Modeling assumptions

Figure 1 shows a simple example of a clustered architecture composed of multiple server nodes with a dispatcher that distributes the incoming requests among the nodes. For this study, we assume that the arriving traffic is dispatched to the nodes according to a round robin strategy. It is also assumed that each node has an associated buffer with limited capacity. Thus, all the requests sent to the node are lost, if its buffer is full. In addition,

the dispatcher runs a monitoring process, e.g., based on heartbeat messages in order to detect node failures. The objective is to early detect the failed nodes and disconnect them from the cluster. So, the cluster supports a failure-detection mechanism that is able to provide a *fail-stop* behavior, i.e. a node crashes and stops working in a way detectable from its neighbors. This is known as the *fail-stop* assumption.

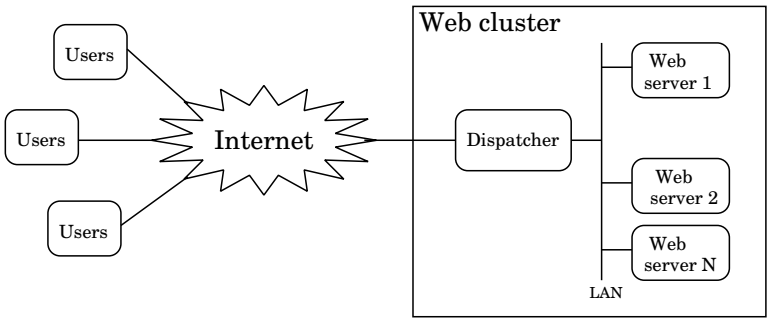

Figure 1. Basic web cluster architecture

In order to evaluate the web service availability, we focus on the server side that is under the direct control of the web designer. We do not include in our analysis the service unavailability caused by failures in the path from the client to the server, or by the failure of the dispatcher. Thus, we take the web designer perspective.

The web service availability is defined as the probability at steady state that requests submitted to the cluster are successfully processed. In other words, they are not lost due to server failures or overloads. We distinguish two main causes of failure:

1. Hardware and software failures that affect the computer hosts running server nodes;
2. Performance-related failures that occur when the incoming requests are not served due to limited capacity of the server buffers.

Failures of nodes have a direct impact on the performance capabilities of the web cluster. Indeed, when nodes fail and are disconnected from the cluster, the remaining healthy nodes have to handle all the original traffic, including the requests previously served by the failed nodes. This leads to an increase of the workload redirected to the remaining nodes, with a potential degradation of the corresponding quality of service. For example, a failure of two nodes in a five node cluster increases by 66% the load to be processed by three remaining nodes. Clearly, an accurate estimation of the web service availability should take into account the performance degradation of the web cluster.

The selection of the appropriate architecture supporting the web service requires knowledge about the recovery strategy following a node failure as well as the consequences of node failures. For this purpose, two recovery strategies are compared, namely: Non Client-Transparent (NCT), and Client-Transparent (CT) (see section 2. for more details).

From the modeling viewpoint, NCT and CT are defined as follows:

- NCT recovery strategy: all requests in progress as well as the input requests directed to the failed node before the failure is detected are lost;

- CT recovery strategy: all requests in progress and the input requests directed to the failed node during failover latency time [1] are not lost; they are redirected to the non failed nodes.

In the following section, we present analytical models addressing these strategies in a unified approach. For CT recovery strategy, we assume that the architecture is implemented as in [10].

4. Cluster Modeling

The evaluation of the web service availability taking into account the modeling assumptions presented in section 3. is carried out adopting a performability approach [11]. The idea consists in combining the results obtained from two models: a performance model and an availability model. The availability model is used to identify the system states that result from the occurrence of failures and recoveries and to evaluate the corresponding steady state probabilities. The performance model takes into account the request arrival and service processes and evaluates performance related measures conditioned on the state of the system as determined from the availability model. This approach is based on the assumption that the system reaches a quasi steady state with respect to the performance-related events, between successive occurrences of failure-recovery events. This assumption is valid when the failure/recovery rates are much lower than the request arrival/service rates, which is typically true in our context.

It is worth to mention that our approach builds on the traditional performability modeling approach [13, 14] by providing closed-form equations for requests loss probability. The requests loss probability can be caused by i) buffer overflow, ii) server failures or, iii) latency of failure detection. The evaluated availability measure allows a distinction among these causes and includes such requests loss probability as a source of web service unavailability.

In the following, we consider a web cluster system composed of N server nodes and a dispatcher balancing the requests among the servers in a round robin way, and capable of detecting the failure of the servers connected to the cluster.

4.1. Availability model

Let us assume that the times between node failures are exponentially distributed with rate γ, and that failure detection occurs with rate α. After detection of a node failure, the cluster is reconfigured by disconnecting the failed node. The latter is reintegrated into the cluster after restoration. The restoration times are assumed to be exponentially distributed with rate τ.

Figure 2 shows the availability model describing the behavior of the cluster governed by server failures, detection, recovery and restoration processes. In states $k = 0, ..., N$, the system has k available nodes capable of processing the input traffic. However, requests could be rejected in these states due to overload conditions. The failure of a server node in state k, leads the cluster to state D_k with a transition rate $k\gamma$. In states D_k, although the server has failed, this failure is not yet perceived by the dispatcher. Accordingly, client requests could still be directed by the dispatcher to the failed node during the

[1] *The failover latency* corresponds to the *detection and recovery time* i.e., the time taken by the dispatcher to detect a failure and remove the failed node from the list of alive nodes in the cluster.

failover latency time. Upon detection, the system moves to state $k-1$ indicating that the number of operational servers has been reduced by one and the restoration of the failed server is initiated. In this model, it is assumed that no other failure can occur when the system is in state D_k. This assumption is acceptable because the failover latency times are generally very small compared to the times to failure.

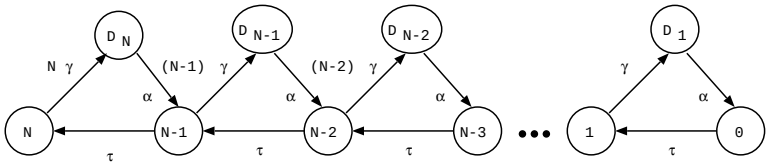

Figure 2. Availability model of the web cluster

We have to solve this model to obtain the steady-state probabilities for states k and D_k, denoted as π_k and π_{D_k}, respectively. Processing is straightforward and the analytical expressions for ($k = 1, ..., N$) are given by equations (1) and (2)

$$\pi_k = \frac{N!}{k!}\left(\frac{\gamma}{\tau}\right)^{N-k} \pi_N, \qquad k = 0 \cdots N \tag{1}$$

$$\pi_{D_k} = \frac{N!}{k!}\left(\frac{\gamma}{\tau}\right)^{N-k} \frac{k\gamma}{\alpha} \pi_N, \quad k = 1 \cdots N \tag{2}$$

where

$$\pi_N = \left[\sum_{j=0}^{N} (\frac{\gamma}{\tau})^j \frac{N!}{(N-j)!} + \sum_{j=0}^{N-1} (\frac{\gamma}{\tau})^j \frac{\gamma(N-j)N!}{\alpha(N-j)!} \right]^{-1}$$

4.2. Performance model

In order to model the cluster input traffic, we consider that the web traffic is characterized by a Poisson process [15], illustrated in Figure 3. The traffic arriving to the web cluster is modeled as a Poisson process with rate λ req/s (requests per second) and is independent of the failure process. Assuming that there are k available servers in the cluster with the input traffic being distributed among the servers, then this system has k independent Poisson arrival processes each one with rate $\lambda' = \frac{\lambda}{k}$.

Also, each server supports a maximum number of requests b (buffer size) and has a service rate of μ req/s. The requests arriving at the server when the buffer is full are rejected.

The performance behavior of each server can be modeled by an M/M/1/b queueing system. Let us denote by p_i the steady state probability of having i requests in this queue. Thus, p_i is calculated as follows (see e.g.,[3])[2]:

[2]Note that the request loss probability due to buffer overflow is given by p_b based on the PASTA (Poisson Arrivals See Time Averages) theorem [20]. Recall that this theorem shows that the probability that a request entering in a queue finds such a queue in a given state is equal to the steady-state probability for this state.

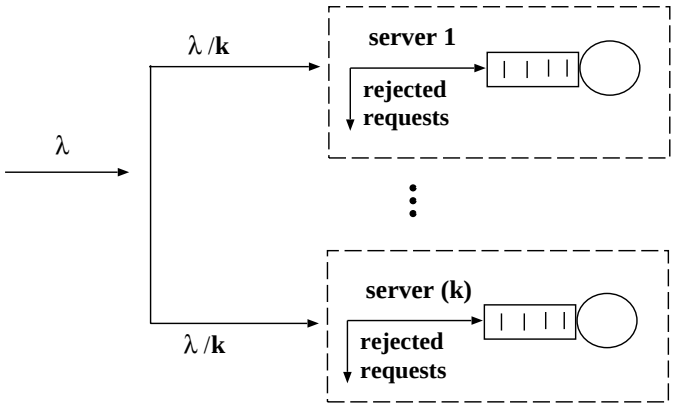

Figure 3. A web cluster with k servers available and load balancing

$$p_i = \begin{cases} \rho^i \frac{1-\rho}{1-\rho^{b+1}} & \text{, if } \lambda' \neq \mu \text{ and } 0 \leq i \leq b \\ \frac{1}{b+1} & \text{, if } \lambda' = \mu \text{ and } 0 \leq i \leq b \end{cases} \tag{3}$$

where ρ is the server load given by $\rho = \frac{\lambda}{k\mu}$.

4.3. Composite Availability - Performance model

The web service availability is evaluated based on a composite availability-performance model. Let us denote by UA, the web service unavailability defined as the probability in steady state that a request is not processed successfully. This means that requests can be lost: i) upon arrival to the cluster, or ii) while they are being processed or waiting for service, due to a server failure, or iii) during failure detection.

In order to evaluate UA, we need to compute the request loss probability in states k and D_k of the availability model (Figure 2), also during the transition between these states.

For $k = 1, \cdots, N$, let us denote by:

- $L(k)$: the request loss probability due to buffer overflow in state k
- $L(k\gamma)$: the request loss probability caused by a transition from state k to state D_k, due to a server failure
- $L(D_k)$: the request loss probability in state D_k, during the node failure detection time.

Web service unavailability UA is computed as follows:

$$UA = \sum_{k=1}^{N} \pi_k \{L(k) + L(k\gamma)\} + \sum_{k=1}^{N} \pi_{D_k} L(D_k) + \pi_0 \tag{4}$$

Recall that π_{D_k}, π_k, and π_0 are the steady state probabilities evaluated from the availability model, given by equations (1) and (2).

The closed analytical equations for $L(k)$, $L(k\gamma)$ and $L(D_k)$, considering both recovery strategies (i.e., NCT and CT) are evaluated in the following sections.

4.3.1. Loss probability due to buffer overflow $L(k)$

Given that k servers are available in the cluster, let us assume that all servers have the same loss probability at steady-state. This assumption holds with a system containing equally utilized servers. Thus, when at least one of the available servers has the buffer full, the probability of loss can be computed as follows:

$$L(k) = [1 - (1 - p_b)^k] \tag{5}$$

Note that p_b is the request loss probability due to buffer overflow. It is obtained from equation (3) with a request arrival rate to each server given by $\lambda' = \frac{\lambda}{k}$. Also, it is worth to mention that $L(k)$ is the same for both recovery strategies.

4.3.2. Loss probability due to server node failure $L(k\gamma)$

This loss scenario occurs only with the NCT recovery strategy. The request loss probability caused by a transition from state k to state D_k corresponds to the loss of all requests (queued or in service), when a web server node fails.

In order to determine the probability that a node failure affects the requests, let $L(k|i)$ be defined as the conditional probability that i requests are lost when a failure occurs. By PASTA theorem [20], the loss probability denoted $L(k\gamma)$ due to server failure is equal to the steady state probability that there are i requests in the queue p_i when a failure occurs, given by

$$L(k\gamma) = \sum_{i=1}^{b} p_i L(k|i)$$

where p_i denotes the steady state probability for an M/M/1/b queue. Let v be defined as $v = \dfrac{\rho\mu}{\mu + k\gamma}$. After simple manipulations, we obtain the following closed-form equation

$$L(k\gamma) = \left[\frac{\rho(1-\rho^b)}{1-\rho^{b+1}}\right] - \left[\frac{(1-\rho)v^2(1-v^b)}{(1-\rho^{b+1})\rho(1-v)}\right] \tag{6}$$

Recall that a M/M/1/b queue system is stable for all values of load $\rho \neq 1$ (see [3]). Therefore, for $\rho = 1$, $L(k\gamma)$ is computed using

$$L(k\gamma) = \left[\frac{b}{b+1}\right] - \left[\frac{(\frac{\mu}{\mu+k\gamma})^2(1-\frac{\mu}{\mu+k\gamma})^b}{(b+1)(1-\frac{\mu}{\mu+k\gamma})}\right] \tag{7}$$

4.3.3. Loss probability during the node failure detection time $L(D_k)$

The computation of request loss probability when the system is in state D_k of the availability model, depends on the recovery strategy. In these states, a node has failed but the

failure was not yet detected by the dispatcher. Therefore, before the system exits from states D_k, i.e. before detection occurs, the dispatcher continues to send requests to the failed node. For both strategies, we need to evaluate the loss probability caused by the failed server and the loss probability caused by the $k-1$ operational servers in the cluster due to their limited buffer capacity. In state D_k, we need to take into account two competing processes: the request arrival process to a server node with associated rate λ', and the failure detection process with associated rate α.

For **NCT recovery strategy**, all the requests sent to the failed node before the failure is detected are lost. The probability of loss in state D_k is given by the probability that one or more arrivals occur in the failed node before failure detection (system exits from state D_k). When the cluster enters in state D_k, only one of two events can occur: 1) detection, which takes the system to state $k-1$, or 2) an arrival, which increases the number of requests in the queue by one. Since these events are assumed to be independent and exponentially distributed with respective means $1/\lambda'$ and $1/\alpha$, then the probability that an arrival happens before detection is given by $\frac{\lambda'}{\lambda'+\alpha}$. As the probability of loss in state D_k corresponds to the probability that at least one arrival occurs, by little law there are λ'/α arrivals on average before detection, which leads to the following equation $\left(\frac{\lambda'}{\lambda'+\alpha}\right)^{\lambda'/\alpha}$.

In addition, the loss probability in state D_k caused by the $k-1$ operational servers due to their finite buffer is $[1-(1-p_b)^{k-1}]$, following arguments explained earlier in the previous section, where p_b is the loss probability for one operational server.

Therefore $L(D_k)$ is computed by the sum of the loss probability of requests sent to the failed node, and the loss probability caused by buffer overflow of the $k-1$ operational nodes in the cluster:

$$L(D_k) = \left(\frac{\lambda'}{\lambda'+\alpha}\right)^{\lambda'/\alpha} + [1-(1-p_b)^{k-1}] \tag{8}$$

For **CT recovery strategy**, the system provides request migration for all user submitted requests, even when a server node fails. Thus, the only loss scenario is due to buffer overflow.

Let us denote by $l(\alpha)$ the probability of loss due to buffer overflow for a ***failed node***. This probability can be derived following the approach proposed in [7] using probability fundamentals. When the cluster enters in state D_k, let us assume that there were i requests in the queue of size b. The loss probability is equal to the probability that $b-i$ arrivals occur before the system exits from state D_k (i.e., transition α takes place). This conditional probability is given by $(\frac{\lambda'}{\lambda'+\alpha})^{b-i}$. By PASTA theorem, the probability that there are i requests in the queue when a detection occurs is equal to the steady state probability that there are i requests in the queue denoted p_i. After some manipulations, the following closed-form equation is obtained

$$l(\alpha) = \left[\frac{1-\rho}{1-\rho^{b+1}}\right]\left[\frac{\lambda'}{\lambda'+\alpha}\right]^b\left[\frac{1-[\frac{\lambda'+\alpha}{\mu}]^{b+1}}{1-[\frac{\lambda'+\alpha}{\mu}]}\right] \tag{9}$$

When $\rho = 1$, $l(\alpha)$ is computed as follows

$$l(\alpha) = \left[\frac{1}{b+1}\right]\left[\frac{\lambda'}{\lambda'+\alpha}\right]^b \left[\frac{1-[\frac{\lambda'+\alpha}{\mu}]^{b+1}}{1-[\frac{\lambda'+\alpha}{\mu}]}\right] \quad (10)$$

We need also to include the loss probability in state D_k caused by the $k-1$ operational servers due to their finite buffer. Note that the operational servers do not stop working during the detection time and we assume that the migrated requests do not increase the loss of requests in the operational servers.

Hence, the loss probability for the CT strategy can be computed by equation:

$$L(D_k) = l(\alpha) + [1-(1-p_b)^{k-1}] \quad (11)$$

5. Evaluation Results

This section presents some results in order to study the sensitivity of the web service availability to different design decisions. For both recovery strategies, we investigate the effect of the cluster size, the impact of the failure detection duration, the reliability of the cluster nodes as well as the overload effects. We quantify the different design decisions analyzing the performance and availability tradeoffs using the web service performability model.

The parameters used in the model may be obtained either via measurements or based on results published in the literature. In particular, the server failure and recovery rates as well as the mean time between request arrivals may be estimated for example through the analysis of the logs maintained by the web server systems.

We first assume that request arrivals to the web cluster follow an exponential distribution [19], where the arrival rate is $\lambda = 20$ req/sec. Table 1 summarizes the nominal values used for performability evaluation, where:

- MTTF: mean time to a node failure;
- MTTD: mean time to detect a node failure;
- MTTR: mean time to node restoration.

Service rate	MTTF	MTTD	MTTR	Buffer size
μ	$1/\gamma$	$1/\alpha$	$1/\tau$	b
5 req/sec	10 days	20 sec.	200 sec.	20

Table 1. Numerical values of the model parameters

The numerical value of $MTTR = 200$ sec. represents essentially failures recovered by reboot and restart of the node.

5.1. Sensitivity to MTTF

Figure 4 shows the web service unavailability UA as a function of the number of servers (cluster size) for two different values of MTTF: 10 days and 4 days. The other values of model parameters are those of Table 1. We analyze clusters with $N = 150$ nodes that is large enough for most of cluster architectures [3].

[3] Although for the sensitivity analysis $N = 150$ is enough to illustrate the main trends of UA, we emphasize that the underlying model allows to evaluate clusters with a larger number of nodes ($N > 150$). Special care has to be taken

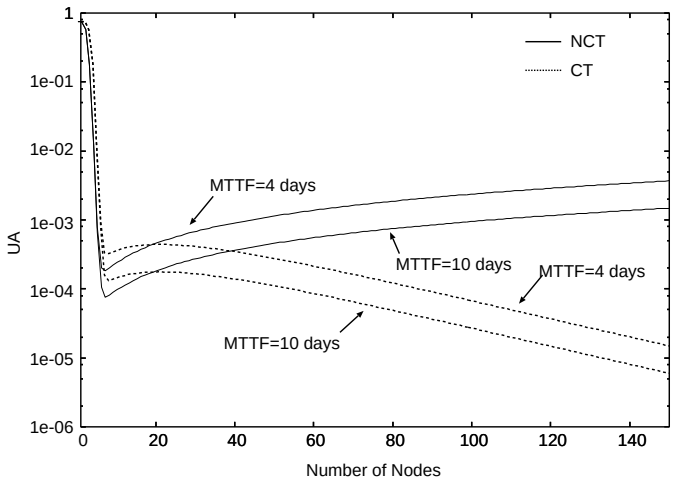

Figure 4. Impact of MTTF on UA

For both strategies, increasing the number of nodes N in the cluster from 1 to 8 leads to a significant availability improvement. However, for a higher number of nodes, we observe different trends:

- **For NCT strategy**, the trend is reversed for $N > 8$. This is explained by the fact that the higher is the number of nodes, the higher is the probability of the system being in states D_k, which increases the probability of requests loss during failure detection time. This fact is more effective when the sejourn time in states D_k is long, i.e. for longer MTTD[4].
- **For CT strategy**, UA increases for a cluster with 9 up to 40 nodes. In this case, the loss probability in states D_k is only due to buffer overflow. This loss probability decreases as the load submitted to each node decreases, which explains why UA restarts to decrease substantially for N beyond 40. In fact, another important reason is that the mean time to detect a failure (MTTD) is not fast enough to avoid the buffer overflow at a failure node, notably for $9 \leq N \leq 40$.

5.2. Sensitivity to MTTD

Figure 5 plots UA as a function of the number of servers. The goal is to analyze the impact of the mean time to detect a failure MTTD, i.e. the mean time spent in states D_k. We consider two different MTTDs: 20 and 2 sec., and two different values of MTTF:10 days and 4 days. The other parameter values are those of Table 1.

The two highest curves of Figure 5 correspond to those of Figure 4. According to these figures, it is clear that there is a notable difference between NCT and CT recovery strategies. In the case of NCT, as the number of nodes increases, UA reaches 10^{-4} for $MTTD = 20$, against $UA \approx 10^{-5}$ for $MTTD = 2$ and $8 \leq N \leq 20$. On the other hand for CT, UA is in the order of 10^{-4} for $MTTD = 20$, and for $MTTD = 2$, UA decreases substantially reaching 10^{-16} as the number of nodes increases to $N = 150$.

for computing the factorial for large N. In this case, we can use the *Stirling approximation*: $N! = \sqrt{2\pi}N^{N+1/2}e^{-N}$.

[4]The time taken to detect a failed node relies on the frequency of heartbeat messages sent to the nodes. Clearly, the higher the frequency, the faster failed nodes will be detected.

This is explained by the fact that when the number of servers increases, the loss probability due to buffer overflow decreases significantly especially for small values of MTTD. However, for NCT, even for small values of MTTD, all the requests that were queued when the server fails as well as those that are directed to this server before failure detection, are lost as well.

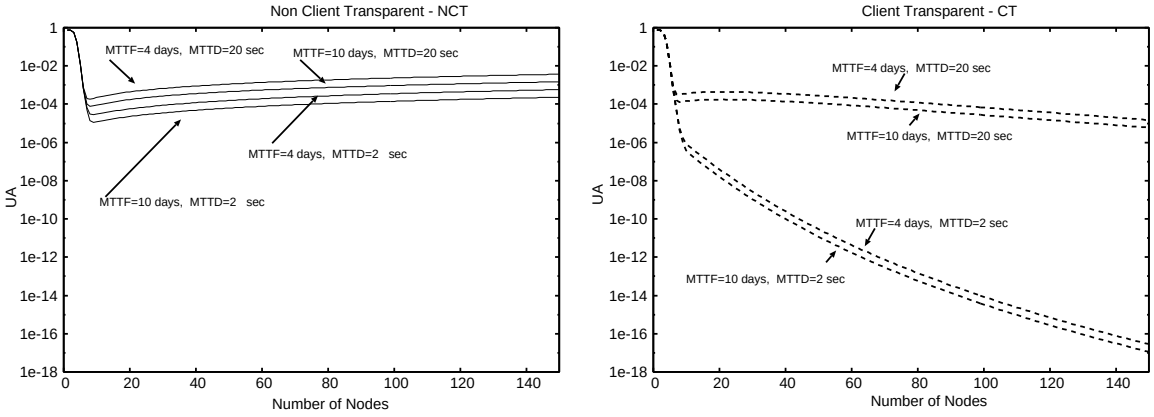

Figure 5. Impact of failure detection duration on UA for both recovery strategies

5.3. Sensitivity to service rate

Figure 6 shows the effect of the service rate on UA, considering two values of μ (5 and 10 req/sec). MTTD is set to 2 sec., while the other parameters are set to their nominal values of Table 1.

In fact, we note that UA decreases faster for a higher service rate as expected, for both strategies. However, the service rate plays a significant role until a certain threshold. According to the figure, the service rate variation does not have effect on UA in a cluster containing more than 10 nodes for both strategies. In contrast, the service rate reduces substantially UA from $N = 1$ to 5 with $\mu = 10$, while UA has been affected from $N = 1$ to 8 with $\mu = 5$.

In this study, the best evaluated UA, **for NCT**, would be obtained for a cluster with $N = 5$ nodes and $\mu = 10$ or with $N = 8$ nodes and $\mu = 5$. However, **for CT**, UA keeps decreasing as the number of nodes increases.

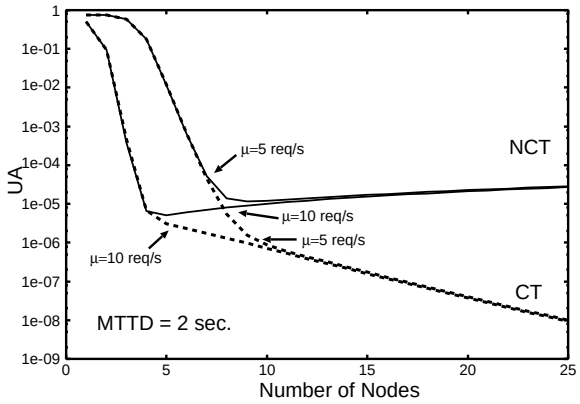

Figure 6. Impact of service rate μ on UA

6. Conclusion

In this paper, we presented performability models for analyzing the availability of web based services implemented in cluster architectures. In particular, we provided closed form equations for web cluster availability taking into account server failures and loss of requests due to system overload.

Sensitivity analyses with respect to cluster characteristics have shown the impact on cluster availability of i) the node failure rate, ii) the failure detection rate, iii) the service rate. These analyses provide useful guidelines for the design of web-based services, since they can be used for dimensioning the web architecture and making the right tradeoffs for achieving high availability with acceptable performance levels.

For instance, we have found that when the number of servers in the cluster is lower than a given threshold (in this study lower than 8), the server processing capacity (service rate plus buffer size) have the highest impact on service availability. In this case, the availability related to the two recovery strategies (i.e., traditional non client transparent strategy and client transparent strategy) is at the same order of magnitude. When the number of servers is higher than this threshold, a significant difference is observed between the two recovery strategies.

The results confirm that the client transparent strategy is better from an availability point of view quantifying this impact. However, the development of a client transparent strategy requires more complex and expensive recovery mechanisms (that allow on-going and arriving requests to be migrated to other servers). Therefore, a tradeoff between the cost of the recovery strategy and its impact on service availability is necessary. Nevertheless, it is worth mentioning that the two recovery strategies are extreme cases. Indeed even in the client transparent strategy some requests may be lost due to imperfect coverage. Considering a coverage factor lower than 100% (i.e. not all the requests are migrated correctly to the non-failed servers), will certainly reduce the difference between the two strategies.

References

[1] N. Aghdaie and Y. Tamir. Client-Transparent Fault-Tolerant Web Service. *IEEE International Performance, Computing, and Communications Conference*, pages 209–216, 2001.

[2] L. Aversa and A. Bertavros. Load balancing a cluster of web servers using distributed packet rewriting. *IEEE International Performance, Computing and Communication Conference*, pages 24–29, 2000.

[3] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains*. John Willey and Sons, Inc., 1998.

[4] E.A. Brewer. Lessons from Giant-Scale Service. *IEEE Internet Computing*, pages 46–55, 2001.

[5] T. Brisco. DNS support for load balancing . *IETF RFC 1794*, 1995.

[6] D. Patterson E. Anderson and E. Brewer. The MagicRouter, an application of fast packet interposing. *http://www.cs.berkeley.edu*, 1996.

[7] S. Garg, Y. Huang, C.M.R. Kintala, K.S. Trivedi, and S. Yajnik. Performance and Reliability Evaluation of Passive Replication Schemes in Application Level Fault Tolerance. *IEEE Dependable Systems and Networks*, 1999.

[8] Cisco Systems Inc. Failover Configuration for LocalDirector. *http://www.cisco.com*, 2000.

[9] M. Kalyanakrishnan, R. K. Iyer, and J. U. Patel. Reliability of Internet Hosts: a Case Study from the End User's Perspective. *Computer Networks*, (31):47–57, 1999.

[10] M. Y. Luo and C. S. Yang. Enabling fault resilience for web services. *Computer Communications*, (25):198–209, 2002.

[11] M. Martinello. Availability Modeling and Evaluation of Web-based Services : A Pragmatic Approach. *PhD thesis LAAS-CNRS*, 2005.

[12] M. Martinello, M. Kaâniche, and K.Kanoun. Web Service Availability : Impact of Error Recovery and Traffic Model. *Journal of Reliability Engineering and System Safety (RESS)*, 89(1):6–16, 2005.

[13] J. F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Journal on Selected Areas in Communications*, 29(8):720–731, 1980.

[14] J. F. Meyer. Closed-form solutions of performability. *IEEE Transactions on Computing*, 7(31):648–657, 1982.

[15] R. Morris and D. Lin. Variance of aggregated web traffic. *IEEE Infocom*, 2000.

[16] K. Nagaraja, G. M. C. Gama, R. Bianchini, R. P. Martin, W. Meira, and T. D. Nguyen. Quantifying the Performability of Cluster-Based Services. *IEEE Transactions on Parallel and Distributed System*, 5(16):456–467, 2005.

[17] D. Oppenheimer and D. A. Patterson. Architecture and Dependability of Large-Scale Internet Services. *IEEE Internet Computing*, pages 41–49, 2002.

[18] M. Colajanni V. Cardellini, E. Casalicchio and P. S. Yu. The state of the art in Locally Distributed Web-Server System. *ACM Computing Surveys*, 34(2):363–371, 2002.

[19] W. Willinger and V. Paxson. Where Mathematics meets the Internet. *Notices of the American Mathematical Society*, 45(8):961–970, 1998.

[20] R. Wolff. Poisson arrivals see time averages. *Operations Research*, (30):223–231, 1982.