Performability Tradeoffs in Active Replication Architectures with Host Failures

Swapna S. Gokhale  B. Dasarathy
Dept. of CSE  Applied Research
Univ. of Connecticut  Telcordia Technologies
Storrs, CT 06269  Piscataway, NJ
ssg@engr.uconn.edu  das@research.telcordia.com

Abstract

Active replication strategy, which involves deploying multiple instances of application components is an effective way of improving the availability of a component-based software system. In this case, when a request needs to be processed by a given component type it is processed simultaneously by all the active instances of that component, which makes it possible to immediately recover from failures. However, simultaneous handling results is a waste of processing capacity and this may have an impact on the performance of the application. A tradeoff arises between the application availability and performance, and this tradeoff will be ultimately determined by the number of instances of each component type and the deployment of these instances across different hardware hosts. In this paper we present a performability analysis methodology for the two commonly used architectural configurations employed in the active replication strategy, namely, the fully overlapping architecture and the partially overlapping architecture. We illustrate the potential of the methodology to conduct quantitative tradeoffs between performance and availability using several examples.
1 Introduction and motivation

Component-based software development consists of developing a software system by assembling it from pre-existing or reusable components rather than building it from scratch. It offers several advantages over the traditional life-cycle software development, the two most notable ones being a reduction in the application development cost and an improvement in the application quality. Cost reduction is enabled by reusing components in multiple applications, whereas quality improvement is achieved by testing components more thoroughly in different setups and environments.

Many component-based software applications are used in life- and mission-critical distributed real-time (DRE) systems. These systems have to meet stringent performance (response time) and availability requirements. Active replication, which consists of deploying multiple instances of each application component is an effective way of guarding against failures and improving availability. In this strategy, when a request needs to be processed by a given type of component, all the active instances of that component type perform the processing. While simultaneous processing of requests by multiple component instances facilitates immediate recovery from failure, it also results in a waste of processing capacity and this may impact the performance of the application. Naturally, a tradeoff arises between application availability and performance and ultimately this tradeoff is influenced by the number of active instances of each component type and the deployment of these instances across different hardware hosts.

Active replication may employ two different types of architectural configurations, namely, the fully overlapping configuration and the partially overlapping configuration. In the fully overlapping configuration, each host has one instance of each component type. In the partially overlapping configuration, each host has only a subset of the application components. These two architectural configurations present different trade-offs and opportunities between performance and availability and such cross-influence of these two attributes is known as performability. In this paper we present a systematic performability analysis methodology for the above two architectural configurations. We illustrate the potential of the methodology to conduct quantitative
performance and availability tradeoffs for the two architectural configurations using several examples.

The remainder of the paper is as follows: Section 2 describes the characteristics of the application under consideration. Section 3 presents the two architectural configurations of the application. Section 4 presents the performability analysis methodology. Section 5 generalizes the performability analysis methodology.

2 Application description

We consider an application with three components, namely A, B and C. We consider two structural arrangements of the three components. These structural arrangements determine how a client request is processed by the application and are as follows:

2.1 Cascade or tiered structure

In the first arrangement, components A, B and C are organized in the form of a cascade. In this case, a client request is processed by the application in the following steps:

1. The client request is received by component A and it processes the request. During the processing it determines that it needs the service of component B. Component A then sends a request to a component B.

2. Component B processes the request from component A and determines that it needs a service from component C. Component B then sends a request to component C.

3. Component C completes processing and sends a response back to component B.

4. Component B completes processing and sends a response back to component A.

5. Component A completes processing and sends a response back to the client.

The processing steps are depicted in the form of a sequence diagram as shown in Figure 1.
2.2 Probabilistic structure

In this structural arrangement, component A invokes the services of either component B or component C (but not both). The services of components $B$ and $C$ are invoked with probabilities $p_B$ and $p_C$ respectively, such that $p_B + p_C = 1.0$. A client request is processed by this structural arrangement in the following steps:

1. The client request is received by component A and it processes the request. During this processing component A determines that it needs the services of either component B or component C. If the services of component C are needed, then it sends a request to component C and the processing continues from step 3. Otherwise, it sends a request to component B.

2. Component B completes processing and sends a response back to component A. The processing continues from step 4.

3. Component C completes processing and sends a response back to component A.

4. Component A completes processing and sends a response back to the client.

The processing steps depicted in the form of a sequence diagram are shown in Figure 2.
The application employs active replication strategy to enhance its availability. There are multiple active instances of each component deployed on three hosts. The management software which is responsible for managing simultaneous processing by multiple component instances resides on a fourth host.

We consider the fully overlapping and partially overlapping architectural configurations. These two configurations differ in terms of the number of instances of each component and the placement of these instances. In the fully overlapping configuration, each component has three instances, with one instance deployed on each host. A pictorial depiction of the fully overlapping architecture is shown in Figure 3. In the partially overlapping architecture, the application has two active instances of each of the components. Each host has one active instance of two out of three components. The partially overlapping architecture is shown in Figure 4.

For both these architectures, the sequence of processing steps of a client request outlined in Section 2 are interspersed with processing by the manager. The updated processing steps for both the task graphs are given in the subsequent subsections.
Figure 3: Fully overlapping architecture

Figure 4: Partially overlapping architecture
3.1 Cascade or tiered structure

The processing steps for the cascade/tiered structure with active replication model are as follows:

M1. Client request is processed by the manager and sent to all the replicas of component A.

1. All the replicas of component A process the client request. During this processing, they determine that they need a service provided by a component B. All the replicas of component A hence send a request to component B.

M2. The requests from all the replicas of component A are processed by the manager to suppress duplicates and a single request from component A is sent to all the replicas of component B.

2. All the replicas of component B process the request from component A. During this processing component B determines that it needs the service provided by component C. As a result, all the replicas of component B send a request to component C.

M3. The requests from all the replicas of component B are processed by the manager to suppress duplicates and a single request from component B is sent to all the replicas of component C.

3. All the replicas of component C process the request from component B and send back a response to component B.

M4. The responses from all the replicas of component C are processed by the manager and a single response is sent back to all the instances of component B.

4. All the replicas of component B process the response from component C and send back a response to component A.

M5. The responses from all the replicas of component B are processed by the manager and a single response is sent back to all the instances of component A.
5. All the replicas of component A complete processing the response from component B and send a response back to the client.

M6. The responses from all the replicas of component type A are processed by the manager to suppress duplicates and a single response is sent back to the client.

3.2 Probabilistic structure

The processing steps involved in fulfilling a client request for the probabilistic structure are as follows:

M1. Client request is processed by the manager and sent to all the replicas of component A.

1. All the replicas of component A process the client request. During this processing, the instances of component A determine that they either need a service provided by component B or by component C. The service offered by component B is needed with probability $p_B$, whereas, the service offered by component C is needed with probability $p_C$. All the replicas of component A hence send a request to component B with probability $p_B$ and component C with probability $p_C$.

M2. The requests from all the replicas of component A are processed by the manager to suppress duplicates and a single request is sent to all the replicas of component B with probability $p_B$ and to component C with probability $p_C$ (continue with step 3).

2. All the replicas of component B process the request from component A and send back a response.

3. All the replicas of component C process the request from component A and send back a response.

M3. The manager processes the response from all the replicas of component B or C and send back a response to component A.

4. All the replicas of component A complete processing the response from components B or C and send a response back to the client.
M4. All the responses from instances of component A are processed by the manager and a single response is sent back to the client.

4 Performability analysis

In this section we analyze the performability of the two alternative architectural configurations described in Section 3. Performability analysis involves analyzing the architectures separately for their availability and performance attributes and then integrating the two attributes to determine the tradeoffs between them.

4.1 Availability analysis

In this section we analyze the two architectures described in Section 3 for their availability attributes. We consider host failures, leaving other types of failures such as object/process failures for future research. Initially, we analyze the availability of a single host, and subsequently, based on the single-host availability we compute the availability of the architecture.

We assume that the failures and the repairs of each host are exponentially distributed with parameters $\lambda$ and $\mu$ respectively. The availability of a host is then given by:

$$A = \frac{\mu}{\lambda + \mu} \quad (1)$$

Based on the availability of a single host, the availability of the application for the two alternative architectures can be obtained as follows.

4.1.1 Fully overlapping architecture

The fully overlapping architecture can withstand the failure of two out of three hosts. Thus, the fully overlapping architecture is available if one or more of the hosts are available, and the application availability in this case, denoted $A_f$ is given by:
$$A_f = \sum_{i=1}^{3} \binom{3}{i} A^i (1 - A)^{3-i}$$

4.1.2 Partially overlapping architecture

The partially overlapping architecture can withstand the failure of only one of the three hosts. Thus, the partially overlapping architecture is available if two or more hosts are available, and the application availability in this case, denoted $A_p$ is given by:

$$A_p = \sum_{i=2}^{3} \binom{3}{i} A^i (1 - A)^{3-i}$$

4.2 Performance analysis

In this section we derive expressions for the response time of the application for the two architectures. We seek to determine the response time of a client request in the steady state, hence we assume that all the instances of all the components are always processing requests. In other words, multiple requests are simultaneously being processed by the application. For example, in the cascade architecture instances of component C could be processing $(i - 2)^{nd}$ request, instances of component B could be processing $(i - 1)^{st}$ request and instances of component A could be processing the $i^{th}$ request. We note that although the application is processing multiple simultaneous requests, all the components may not always be processing requests. This may occur, especially if there is a large difference in the processing times of the components. However, by assuming that all the instances of all the components are always processing requests, a pessimistic or an upper bound on the response time may be obtained.

For the purpose of performance analysis, we introduce the following random variables to denote the execution times of the components:

- $T_{A,1}$ – Execution time of component A until it determines that it needs a service provided by compo-
nent B in the fully overlapping architecture and components B/C in the partially overlapping architecture (Step #1 in Figures 1 and in Figure 2.)

- $T_{A,2}$ – Execution time of component A, after it receives response from component B in the fully overlapping architecture and components B or C in the partially overlapping architecture (Step #5 in Figure 1 and step #4 in Figure 2).

- $T_{B,1}$ – Execution time of component B after it receives a request from component A until it determines that it needs a service from component C in the fully overlapping architecture (Step #2 in Figure 1).

- $T_{B,2}$ – Execution time of component B after it receives a request from component C in the fully overlapping architecture (Step #4 in Figure 1).

- $T_B$ – Execution time of component B after it receives a request from component A in the partially overlapping architecture (Step #2 in Figure 2).

- $T_C$ – Execution time of component C after it receives a request from component B in the fully overlapping architecture and component A in the partially overlapping architecture (Step #3 in Figures 1).

- $T_m$ – Processing time of the manager.

We let $\alpha$ denote the fraction of the processing capacity of each host consumed in sending and receiving messages from the manager. The actual execution of the components on the hosts will thus be slowed down by a factor of $(1 - \alpha)$.

### 4.2.1 Fully overlapping architecture

In the fully overlapping architecture, each host has three components executing simultaneously. Thus, the processing time of each component will be thrice as much as the processing time if only a single component instance were to execute on a host at a given time. The end-to-end response time for the cascade task graph, denoted $E[R_{f,c}]$ is given by:
\[ E[R_{f,c}] = 6E[T_m] + 3E\left[ \frac{T_{A,1}}{1 - \alpha} \right] + 3E\left[ \frac{T_{A,2}}{1 - \alpha} \right] + 3E\left[ \frac{T_{B,1}}{1 - \alpha} \right] + 3E\left[ \frac{T_{B,2}}{1 - \alpha} \right] + 3E\left[ \frac{T_C}{1 - \alpha} \right] \] 

(4)

The end-to-end response time for the probabilistic task graph, denoted \( E[R_{f,p}] \) is given by:

\[ E[R_{f,p}] = 4E[T_m] + 3E\left[ \frac{T_{A,1}}{1 - \alpha} \right] + 3E\left[ \frac{T_{A,2}}{1 - \alpha} \right] + 3p_B E\left[ \frac{T_B}{1 - \alpha} \right] + 3p_C E\left[ \frac{T_C}{1 - \alpha} \right] \] 

(5)

4.2.2 Partially overlapping architecture

In the partially overlapping architecture, since each host has only two instances, each component takes twice as long to process a particular request compared to the processing time had only one component resided on a single host. Thus, in this case the expected end-to-end response time of the cascade and probabilistic task graphs, denoted \( E[R_{p,c}] \) and \( E[R_{p,p}] \) is given by:

\[ E[R_{p,c}] = 6E[T_m] + 2E\left[ \frac{T_{A,1}}{1 - \alpha} \right] + 2E\left[ \frac{T_{A,2}}{1 - \alpha} \right] + 2E\left[ \frac{T_{B,1}}{1 - \alpha} \right] + 2E\left[ \frac{T_{B,2}}{1 - \alpha} \right] + 2E\left[ \frac{T_C}{1 - \alpha} \right] \] 

(6)

\[ E[R_{p,p}] = 4E[T_m] + 2E\left[ \frac{T_{A,1}}{1 - \alpha} \right] + 2E\left[ \frac{T_{A,2}}{1 - \alpha} \right] + 2p_B E\left[ \frac{T_B}{1 - \alpha} \right] + 2p_C E\left[ \frac{T_C}{1 - \alpha} \right] \] 

(7)

4.3 Performability tradeoffs

The performance and availability analysis reported in Sections 4.1 and 4.2 clearly reveal the tradeoffs among these two attributes. The fully overlapping architecture exhibits higher availability since it can withstand two processor failures. However, the price for high availability is lower performance or higher end-to-end response time. On the other hand, the partially overlapping architecture has higher performance but lower availability, since it can withstand the failure of only one processor. The equations derived in the above two sections can be easily used to conduct such tradeoffs for different parameter values. A systematic,
quantitative analysis of these tradeoffs can guide in the selection of an appropriate configuration; dependent on the performance and availability targets of the application domain.

5 Generalized methodology

In this section we generalize the performability analysis methodology presented in Section 4 for an application consisting of \( o \) components. The application is to be deployed on \( n \) hosts. In the fully overlapping architecture, each one of the \( n \) hosts has one instance each of the \( o \) components. In the partially overlapping architecture, each one of the \( o \) components has a total of \( m \) instances such that \( m < n \). Further, these \( m \) instances of \( o \) components are deployed in such a way that there is only one instance of each component on a given host and each host has \( q \) component instances.

5.1 Availability analysis

The fully overlapping architecture can withstand \( n - 1 \) failures, and hence its availability is given by:

\[
A_f = \binom{n}{n-1} (A)(1-A)^{n-1}
\]

In the partially overlapping architecture, since each component has \( m \) instances deployed on \( m \) out of \( n \) hosts, this architecture can withstand \( m - 1 \) failures. Thus, the availability of this architecture is given by:

\[
A_p = \binom{n}{m-1} (A)^{(n-m+1)}(1-A)^{m-1}
\]

5.2 Performance analysis

The execution of the components is slowed by a factor of \( 1/o \) in the fully overlapping architecture and a factor of \( 1/q \) in the partially overlapping architecture. Also, if the number of component invocations is \( c \),
then the time incurred in processing by the manager software is $2cE[T_m]$. The end-to-end response time will depend on the actual task structure of the application components.

5.3 Performability analysis

Thus, the fully overlapping architecture has high availability but low performance. The partially overlapping architecture has lower availability but better performance than the fully overlapping architecture. The level of tradeoff between performance and availability in the partially overlapping architecture will be determined by the parameters $n, m, o$ and $q$. 