Failure Detection and Recovery Modelling for Multi-layered Service Systems

Olivia Das, C. Murray Woodside
Dept. of Systems and Computer Engineering
Carleton University, Ottawa, Canada
email: odas@sce.carleton.ca, cmw@sce.carleton.ca
Date: June 22, 2001

Abstract

Existing approaches to combined performance and availability modelling of multi-layered service systems do not consider the delays associated with the detection of failure (repair) and subsequent reconfiguration. The detection delay influences the runtime overhead while both of them together affects the system downtime. The effect of these two types of delays should therefore be considered for precise modelling of the system. This position paper describes the avenues for extending a previous model on layered systems by considering detection delay incurred due to the propagation of failure or repair information through the layers and the delay induced due to system reconfiguration, thereby enhancing the modelling accuracy.

1. Introduction

Distributed software systems are usually structured in layers with some kind of user-interface tasks as the topmost layer, making requests to various layers of servers. Client server systems and Open distributed processing systems such as DCE, ANSA and CORBA are structured this way. [Woodside96] introduced an elegant approach to express the layered failure and repair dependencies while [Das98, Das01] provided an efficient algorithm for identifying equivalent system states from performance viewpoint, in these systems. However, the work done in [Woodside96, Das98, Das01] is limited to instantaneous perfect detection and reconfiguration, and independent failures and repairs.

This position paper describes avenues for extending the previous work on layered systems, by considering:

- a heartbeat mechanism for detecting failures (repairs) of processes and processors,
- an architecture for reporting on operational status (failures and repairs), with its own delays and dependencies,
- delays for detection and reconfiguration

Initially the attention will remain focussed on crash-stop failures [Schneider93]. Related to failure detection, it is assumed that every host in the system runs a single status detector process. A status detector receives heartbeats from the locally running processes in order to detect the process failures and it exchanges heartbeats with other status detectors in order to detect the processor failures. Similar heartbeat-based mechanism for detecting failures of processes or processors has already been employed in systems like Tandem's NCAPS [Laranjeira98], NT-Swift [Huang98] and Globus toolkit [Stelling98].

2. Layered Models capturing Failure Occurrence and Repair Behavior

Figure 1 shows a typical example of a layered model proposed in [Woodside96, Das98, Das01] with one client task, Task 1, at the top and three layers of servers, Task 2-5, the intermediate layers and Task 6, the bottom layer. An arrow represents a request-reply interaction, such as an RPC. Processors are represented by ovals. Alternative targets for the service, Service 1, needed by Task 2 are Task 3-5 with an order of preference. Figure 1 describes both the architecture of the software and its configuration. Failure and repair rates are provided for each component (either a task or a processor) as availability parameters. The performance parameters attached to such model are the mean total demand for execution on the processor and the mean number of requests for each interaction.

In a layered model, there are two kinds of components that can undergo failure: tasks and processors. Each component has its own structure state s_i (0 for failed and 1 for operational). Considering that there are N components in the system, the *system configuration state* S [Das98] is defined as the vector of structure states of components, given by $S = (s_1, s_2, ..., s_N)$. Because of the layered structure of such models, more than one system configuration state may correspond to a single operational configuration of the system from the performance viewpoint [Das98].

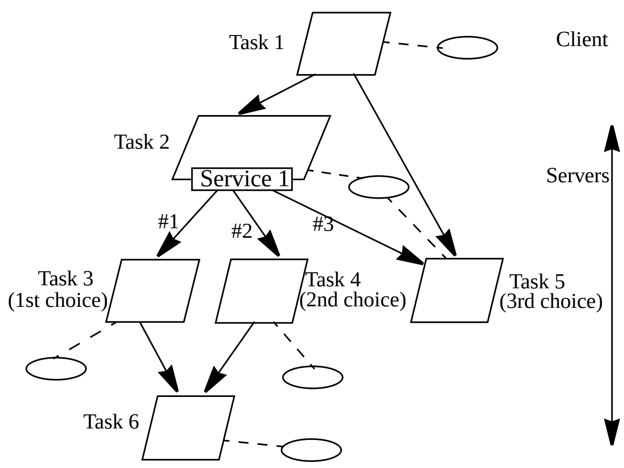


Fig. 1 A Layered model [Das98] with three layers. Alternative targets for Service1 are Tasks 3-5 in increasing order of preference.

In order to model the effect of delay associated with failure (repair) detection, we need a detection architecture in addition to the software architecture. The next section discusses the detection architecture used in this work.

3. Detection Architecture

Each host runs a status detector (SD) task. An SD receives periodic status messages (heartbeats) from the locally running tasks in order to detect task failures (repairs). It exchanges heartbeats with other status detectors in order to detect processor failures (repairs) and to propagate information about a failure (repair). The exchange of heartbeats among SDs is based on the following assumption: if a task (primary or backup) provide service to another remote task, then the local SD of service provider task would send heartbeats to the SD that is local to the service requester task. Figure 2 shows the detection architecture for Figure 1. An SD detects a failure based on a missing heartbeat. An SD receiving heartbeats from a remote SD cannot distinguish between remote SD's failure or its associated host failure. Therefore, failure of an SD is interpreted as the corresponding host failure.

Upon detecting the failure of a task, the local SD attempts to restart it on the same host up to a threshold number of times within a given interval as part of local recovery. If the threshold is exceeded within that interval, it propagates the knowledge about the failure to the SDs of upper layers, which might result in the system switching from primary targets to backups. Similar propagation of information occurs after a repair, that might result in the system switching back from backups to primaries. In case of failure of a processor or its associated SD, the recovery process is initiated by the remote client SDs. However when a processor or its associated SD is repaired, the rejoining process involves obtaining the status information from the lower layers in order to evaluate the resulting status of the services it provides and then propagating that information to the upper layers through the heartbeats.

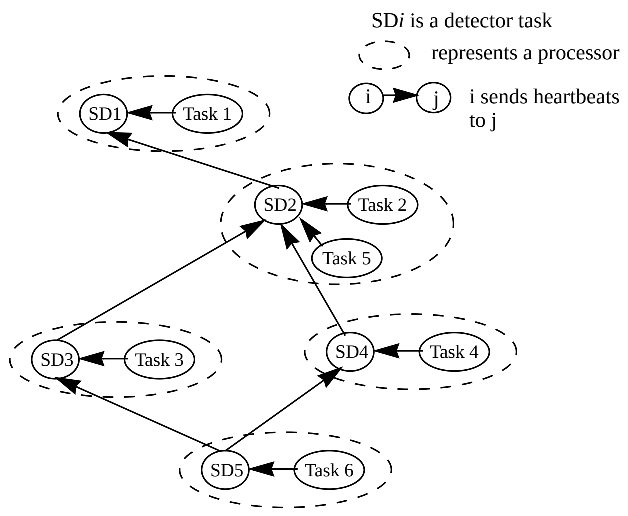


Fig. 2 Detection Architecture for Figure 1.

4. Expressing Detection and Reconfiguration

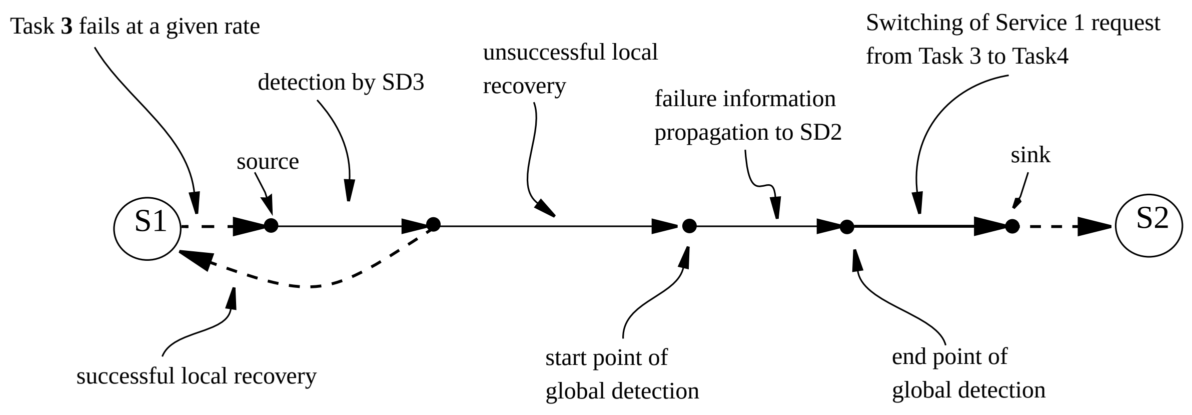
The detection and reconfiguration parameters to be provided in the model are as follows:

- delay of detection propagation from one task to another. It can be found from the heartbeat interval.
- restart delay of each task
- reconfiguration delay for each service request that has alternative targets.
- probability of successful local recovery of a task, within the given interval.

Given these parameters, we could define a directed graph that describes the sequence of detection and reconfiguration actions performed by the system in response to a failure (repair). It can be obtained using the software architecture, detection architecture and the state of the system at the time of failure (repair). We term this graph as the *detection-reconfiguration* graph. A sample illustration of this graph is provided in Figure 3. Each arc in this graph represents an action and could be labeled with appropriate rates. The illustration in Figure 3 represents one detection path. This graph can also have multiple concurrent detection paths originating from the source of failure (repair). In that case, the actions along all

S1: system configuration state at the time of failure of Task 3

S2: system configuration state after the system reconfiguration is complete

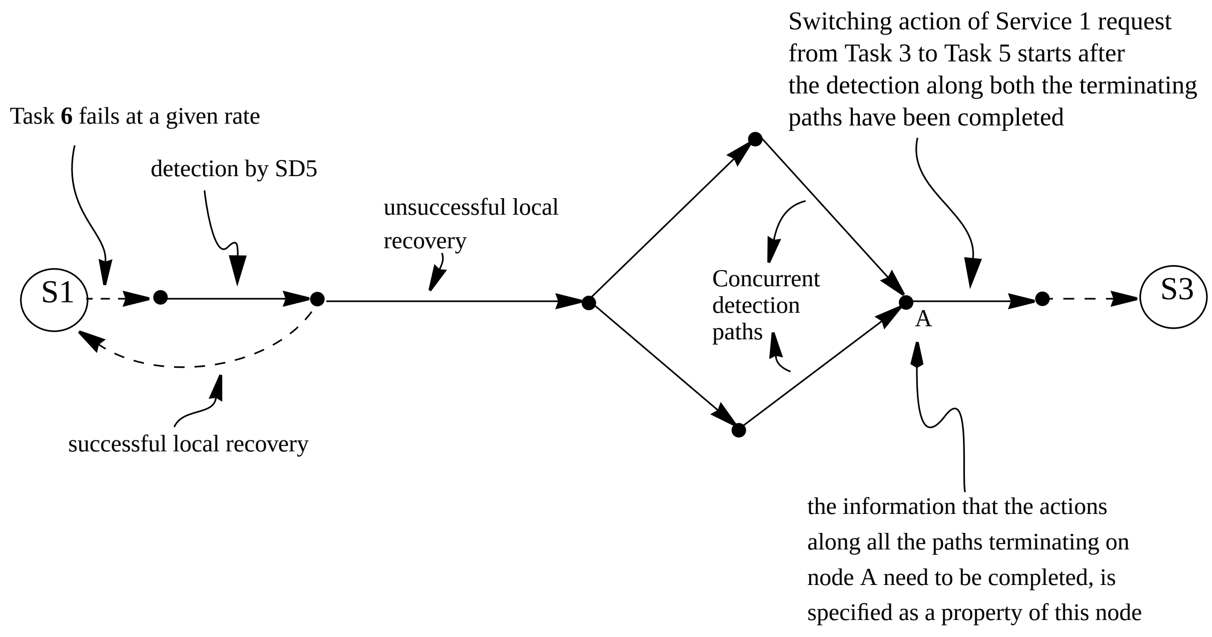


Task **3** fails at a given rate
detection by SD3
source
unsuccessful local recovery
failure information propagation to SD2
Switching of Service 1 request from Task 3 to Task4
sink
S1
S2
successful local recovery
start point of global detection
end point of global detection

Figure 3. *Detection-Reconfiguration* graph for the system in Figure 1 in response to failure of Task 3 where all the components were operational at the time of failure.

S1: system configuration state at the time of failure of Task 6

S3: system configuration state after the system reconfiguration is complete



Switching action of Service 1 request from Task 3 to Task 5 starts after the detection along both the terminating paths have been completed
Task **6** fails at a given rate
detection by SD5
unsuccessful local recovery
Concurrent detection paths
S1
A
S3
successful local recovery
the information that the actions along all the paths terminating on node A need to be completed, is specified as a property of this node

Figure 4. *Detection-Reconfiguration* graph for the system in Figure 1 in response to failure of Task 6 where all the components were operational at the time of failure.

the paths need to be completed before the system can go to the next system configuration state. Whenever multiple detection paths terminate on a node, a property on that node is specified which determines the path(s) needed to be traversed before pursuing the next action(s), if any. Figure 4 illustrates an example of multiple detection-paths terminating at a node, A. It describes the actions performed by the system in Figure 1 in response to the failure of Task 6 where all the components were operational at the time of failure. In this case, the arc emanating from node A represents a switching action of Service 1 request (in Task 2) from Task 3 to Task 5. This action can only commence after the failure of both Task 3 and Task 4 has been detected by the detector, SD2. This information is captured as a property of node A.

The detection-reconfiguration graph enables us to conveniently visualize the concurrent detection and reconfiguration behavior of the system. We utilize this graph to generate the set of states the system goes through in response to a failure (repair).

The runtime overhead due to sending and receiving of heartbeats in the system is modelled by adding a overhead task in each processor. An overhead task is not added in a processor where only backup tasks are running since such tasks do not contribute in providing services. Although these backup tasks are involved in sending/receiving heartbeats, the overhead on the processor due to this activity is negligible and can be ignored. This assumption helped us in reducing the number of distinct performance models.

5. Model Solution

We plan to obtain the desired performability measures for the layered model as follows:

1. Determine all the distinct operational configurations of the system using the algorithm of [Das98]. In each configuration, add an overhead task on each processor representing the performance overhead due to sending and receiving of heartbeats. Apply the layered queueing analysis [Woodside89, Woodside95] to obtain the reward rates for each operational configuration.

2. Find the Markov chain that include the reachable set of system configuration states S. It incorporates the detection-reconfiguration submodel in between every two system configuration states.

3. Associate the reward rate to each state of the resulting Markov chain based on the operational configuration it corresponds to.

4. Solve the resulting Markov reward model to obtain the desired measures.

6. Conclusion

An approach has been described for including detection and reconfiguration in a layered performability model. The key question to be answered is the complexity of solution for the probabilities of different operational configurations.

7. References

[Schneider93] F. B. Schneider, What Good are Models and What Models are Good, in Sape Mullender, Editor, Distributed Systems, ACM Press, 1993.

[Das98] O. Das and C. M. Woodside, The Fault-Tolerant Layered Queueing Network Model for Performability of Distributed Systems, IEEE International Computer Performance and Dependability Symposium (IPDS'98), Sept. 1998, pp. 132-141.

[Woodside96] C. M. Woodside, Performability Modelling for multi-layered Service Systems, 3rd International Workshop on Performability of Computer and Communication Systems, Bloomingdale, Illinois, Sept. 1996.

[Das01] O. Das and C. M. Woodside, Evaluating Layered Distributed Software Systems with Fault-Tolerant Features, Performance Evaluation, vol. 45, May 2001, pp. 57-76.

[Huang98] Y. Huang, P. Y. Chung, C. M. R. Kintala, D. Liang and C. Wang, NT-SwiFT: Software implemented Fault-Tolerance for Windows-NT, Proceedings of 2nd USENIX WindowsNT Symposium, Aug. 3-5, 1998.

[Laranjeira98] L. A. Laranjeira, NCAPS: Application High Availability in Unix Computer Clusters, Proc. of 28th Intl. Symposium on Fault Tolerant Computing (FTCS-28), June 1998, pp. 441-450.

[Stelling98] P. Stelling, I. Foster, C. Kesselman, C. Lee and G. von Laszewski, A Fault Detection Service for Wide Area Distributed Computations, in Proc. of 7th IEEE Symposium on High Performance Distributed Computations, 1998, pp. 268-278.

[Woodside89] C. M. Woodside, Throughput Calculation for Basic Stochastic Rendezvous Networks, Performance Evaluation, vol. 9, no. 2, April 1989, pp. 143-160.

[Woodside95] C. M. Woodside, J. E. Neilson, D. C. Petriu and S. Majumdar, The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software, IEEE Trans. on Computers, vol. 44, no. 1, January 1995, pp. 20-34.