Enhancing Robustness of Telecommunications Networks

MOSHE ZVIRAN
Faculty of Management
Tel Aviv University
Tel Aviv, ISRAEL

CHANAN GLEZER
Department of Information Systems Engineering
Ben Gurion University of the Negev
Beer Sheva, ISRAEL 84105

Abstract: - This article presents an algorithmic reconfiguration model, combining mechanisms of load balancing and fault tolerance in order to increase utilization of computer resources in a distributed multi-server, multi-tasking environment. The model has been empirically tested in a network of computers controlling telecommunication hubs and is compared to previous efforts to address this challenge.

Key-Words: - computer networks, dependability, fault tolerance, load balancing.

1. Introduction

Telecommunication systems as well as other mission-critical systems such as utility, banking, medical, military and transportation networks rely heavily on state-of-the-art computing and telecommunication technologies [8]. The power distribution and telecommunications infrastructure have been considered robust against simple, mundane failures. Nevertheless, they cannot tolerate coordinated malfunctions on a much larger scale such as the one exhibited during the September 11th attacks and the massive power outage across North America during August 2003. In order to address substantial threats, as those mentioned above, there is a pressing need to enhance the robustness of telecommunications networks by incorporating prominent computerized mechanisms. This article proposes a mechanism that enables better protection and utilization of a telecommunication network by exploiting prominent fault tolerance and load balancing mechanisms.

The issue of fault tolerance in computerized networks has received much attention both in academia and industry. Architectures for fault-tolerant systems are currently being implemented at both the hardware and software levels. Most of these implementations adopt a hot-standby approach [2], which is based on duplication of computer resources using check-pointing and message-logging techniques [4]. Under this approach, whenever a main computer becomes inoperative, a similar standby computer is immediately activated to replace the inoperative computer. After the inoperative computer has recovered or been fixed it is reinstated and the temporary hot-standby computer is inactivated.

Load Balancing in a Distributed Computing System (DCS) [10] refers to allocating and independently performing computation tasks across a heterogeneous network of processors. Load balancing mechanisms may be categorized as static or dynamic. Static load balancing mechanisms assume all information about the composition of the group of scheduled tasks is available before processing begins. Dynamic load balancing mechanisms, on the other hand, do not use such a priori information. Each task is introduced to the system at one of the available processors and the task's characteristics are then evaluated to determine whether the task should stay at that processor or migrate to another processor. Nevertheless, only a small number of relatively simple load balancing algorithms have been actually applied in practice.

Several experiences have been reported on combining load-sharing and fault-tolerance mechanisms, (e.g., Remote Execution Manager [9], Condor [7], and DAWGS [3], Coterie [10]). Nevertheless, these systems exhibit only limited fault tolerance capabilities. The most comprehensive attempt to constructs a reconfigurable system was made in GATOSTAR [4]. The work of Folliot and Sens on GATOSTAR focused on the architectural design of the system, prototype implementation and empirical evaluation.

The goal of this article is to develop, illustrate and practically evaluate an algorithmic model that combines load sharing and hot standby using the Hamilton method [5], which was originally proposed for apportioning election results in the US House of Representatives. The description of the model focuses on the task redistribution mechanism, and it is evaluated in a real-life setting by checking its ability to effectively control a set of digital hubs that enable national telecommunication. Based on this evaluation, conclusions are drawn regarding the feasibility of the model.

2. The Reconfiguration Model

The proposed model is based on combining the mechanisms for fault tolerance and load balancing in a multi-server and multi-tasking computer network. Following is a description of the assumptions underlying the model.

1) Each computer connected to the network can process several types of tasks concurrently based on the unique requirements of each task.
2) The tasks are processed from queues by (expert) servers operating under the computers connected to the network. 
3) In case one of the servers becomes inoperative, the tasks in its incoming queue are routed to similar servers running concurrently on different computers. 
4) Servers of a given type on different computers may have a different processing capacity. 
5) The prototype derived from the conceptual model should accommodate safety mechanisms that will enable it to handle both crash-type and arbitrary (Byzantine) failures, resulting in a higher failure mode coverage [6]. See details on the implementation protocols (section 4).

The complexity of fault tolerance in the scenario described above stems from the dynamic and uncertain nature of the network (e.g., computers can be installed or removed in real time, unexpected software/hardware crashes may occur). It is the need to provide end-users with quality service at a minimum level of response time that prompts the seeking and evaluation of mechanisms that will detect faults as well as rapidly adjust the performance of the network so that the desired quality standards are maintained. Effective synchronization and communication protocols are critical for the success of such a system.

The reconfiguration model is algorithmic and comprises the following elements.

- **Network Status**: A set of vectors and matrices that capture the actual state of the network at any given point in time (termed *logical configuration*). These elements describe which servers and computers are active and which tasks are processed on each server at any point in time. It also includes operational instructions on what to do with the tasks running on a server in case the host computer becomes inoperative.

- **Task-Reconfiguration Algorithm**: An algorithmic set of procedures that transform the network status elements so that they capture and react to changes in the state of the network (termed *events*) with minimal delay.

Note that contrary to the logical configuration, the *physical configuration* of the network refers to hardware features (e.g., memory/CPU power, I/O devices). Changes in the physical configuration are less frequent than changes in the logical configuration. The former fall outside the focus of the model because they cannot affect the behavior of the model unless they are first reflected in the logical configuration (e.g., register a new computer in an appropriate matrix).

Figure 1 depicts an example of a network to which the model may be usefully applied. The network is composed of three computers A, B, C, and each computer has three types of servers, which handle tasks of type a, b, c. An example the different tasks could be controlling different manufacturers’ models of communication hubs in a telecommunication network.

For each type of task, the tasks processed by the network are numerated sequentially. As an example, currently computer A is processing tasks a1, a2 via server Aa; computer B is processing task a3 via server Ba; and computer C is processing tasks a4 via server Ca. Computer C has only two operative servers (Ca, Cb) and is not handling tasks of type c (server Cc is inoperative).

The basic principle of the model is to dynamically redistribute tasks between servers in the network in response to events. When a negative event occurs in the network (e.g., a computer crash, or arbitrary failure), the model reallocates active tasks on the stalled computer to other available computers according to a proportional ratio determined by the relative importance of the servers. The importance (vote) of a server is based on the system manager’s evaluation of the relative processing capacity among all servers of a give type (on different computers). In case there is a leftover task as a result of the above event, then this task is allocated to the computer that has the highest remainder, using the Hamilton method [5].

This approach can be applied to the event of system initialization as well. Following is a description of the elements used in the reconfiguration model

2.1 **Network Status**: The time-variant configuration of the network is captured by a set of vectors and matrices. These elements are further divided into elements that capture the state of the network at a given point in time, elements that
capture the base configuration of the network as specified by the system manager. The latter can only be changed by the system's administrator, whereas the former change as a result of events that occur in the network (e.g., a power malfunction affecting a group of computers in the network). Following is a list of vectors and matrices used to describe the logical configuration of the network. Type of values permitted in the following matrices and vectors is denoted in square brackets.

\[ N_{Bi} \] - Initial operation state of computers [Boolean: \( N_{Bi}=1 \)~ i-th computer is active at system initialization; otherwise \( N_{Bi}=0 \)].

\[ N_i \] - Current operation state of computers [Boolean: \( N_i=1 \)~ i-th computer is active at the moment; otherwise \( N_i=0 \)].

\[ S_{Bij} \] - Initial operation state of expert servers on computers (modified only by the system's administrator) [Boolean: \( S_{Bij}=1 \)~ the j-th server on the i-th computer is active at system initialization; otherwise \( S_{Bij}=0 \)].

\[ S_{ij} \] - Current operation state of expert servers on computers [Boolean: \( S_{ij}=1 \)~ the j-th server on the i-th computer is active at the moment; otherwise \( S_{ij}=0 \)].

\[ V_{Bij} \] - relative vote (importance) of a given type of server on different computers. Importance is determined by the system administrator based on processing capacity [Real].

\[ U_j \] - number of relevant votes of task type j (running on all computers at the moment) [Integer].

\[ V_{ij} \] - normalized relative vote of a given type of server on different computers. \( (N_i \times S_{ij} \times V_{Bij})/U_j \) [Real].

\[ C_{Bjk} \] - status of different tasks processed by expert servers on different computers [Boolean: \( C_{Bjk}=1 \)~ task k of type j is currently processed; otherwise \( C_{Bjk}=0 \)]

\[ Q_{ij} \] - number of tasks processed by expert servers on different computers [Integer: \( Q_{ij}=q \) denotes that server j on i-th computer is handling q tasks of its type at the moment]

\[ M_{ijk} \] - allocation of tasks to different servers on different computers. [Boolean: \( M_{ijk}=1 \) denotes that k-th task of type j is currently being processed by j-th server on i-th computer; otherwise \( M_{ijk}=0 \)].

\[ W_j \] - total number of tasks processed by servers of type j [Integer: \( W_j=n \) denotes that there are n tasks currently being processed by all servers of types j].

\[ Y_j \] - total number of tasks currently allocated to servers of type j [Integer: \( Y_j=n \) denotes that there are n tasks already allocated to all servers of types j].

\[ W_j-Y_j \] - number of tasks of type j that still need to be allocated [Integer].

\[ R_{ij} \] - "remainder" of tasks to be allocated [\( R_{ij}=n \)~ there are n tasks to be allocated to j-th server on i-th computer].

\[ M_{OLDijk} \] - old allocation of tasks to different servers on different computers [Boolean: mirror of \( M_{ijk} \) before reallocation of tasks to servers as a result of events in the network].

\[ H_{ijk} \] - "held tasks" (operational matrix) = \( M_{OLDijk} \& M_{ijk} \) [Boolean: \( H_{ijk}=1 \) denotes that k-th task of j-th type will continue to be processed on j-th server on i-th computer as a result of the current event].

\[ G_{ijk} \] - "Get tasks" (operational matrix) = \( M_{ijk} \& \neg H_{ijk} \) [Boolean: \( G_{ijk}=1 \) denotes that k-th task of j-th type will be activated on j-th server on i-th computer as a result of the current event].

\[ F_{ijk} \] - "Free tasks" (operational matrix) = \( M_{OLDijk} \& H_{ijk} \) [Boolean: \( F_{ijk}=1 \)~ k-th task of j-th type will be not continue to be processed on j-th server on i-th computer as a result of the current event].

\[ MON_{ijk} \] - performance monitor of tasks [Character: \( MON_{ijk} = \text{state} \)~ current state of k-th task on j-th server running of i-th computer]. Note: "state" is domain-specific knowledge used to provide feedback about the system to end-users.

2.2 Task Reconfiguration Algorithm: A set of procedures (methods) that operate on the network status in response to events that occur in the network such as initialization and malfunctions. Figure 2 presents the structure of the reconfiguration algorithm, in charge of tasks' redistribution among servers and computers. It explains the role of each encapsulated procedure and the calling sequence among the procedures. The driver function is called \textit{Deal_Missions} and is in charge of invoking the whole redistribution algorithm. The algorithm uses the relative vote of each server to allocate a discrete number of tasks of each type among different computers. After this is done, the remaining tasks for every task type are allocated using the Hamilton method [5].
The Hamilton method provides a solution to the apportionment problem\footnote{cw.prenhall.com/book bind/pubbooks/tannenbaum}, which is all about dividing up items so that their sum is maintained. The items can be seats in a parliament, section offerings of courses, or, as in our case, tasks running on servers. The Hamilton method was proposed during the late 18\textsuperscript{th} century by Alexander Hamilton when the US house of representatives was first apportioned.

The main goal of Hamilton was to ensure a fair and balanced representation of the people's will. The method awards leftovers according to the "largest fraction"; however, it was vetoed by President Washington and several other methods have since been used.

The main limitation of Hamilton's method is that when using it over time and after an increase in the overall items to be apportioned (e.g., a new seat to be allocated), some states might actually lose a seat even though their population has not changed (e.g., the Alabama, population, and new states paradoxes).

We decided to adopt the Hamilton method for reallocation of tasks among servers because it is a simple, straightforward and useful method and also because the biases mentioned above are too rare to invalidate the method.

In our case, the remainders for every task type are calculated using the relative votes of all computers for the task. The remainders are then sorted in descending order and the computer that has the highest remainder is granted the first unallocated task. The rest of the tasks are allocated to the other computers by descending order of the remainders.

### 3. Evaluation of the Model

The proposed reconfiguration model was evaluated on a large national digital telecommunications network comprising approximately 200 hubs of the following types: TX-1, TMX-10, and TMX-100 (manufactured by Northern Telecom) and System-12 (manufactured by Alcatel) serving between 1000 and 20,000 customers each. As an example, the System-12 hub is a complex hardware /software device that runs several tens of modules concurrently. The modules are responsible for various tasks (e.g., central control, connection with customers, message routing, connection bus with other hubs, distribution control). The System-12 hub uses approximately 100 types of status messages in order to monitor and coordinate the operation of the hub (e.g., detecting and handling malfunctions).

The model for controlling the network was implemented using the C programming language. The system operates over the VAX/OpenVMS operating system running on two VAX 4000-5000 computers and using the Digital
The benefit from using the proposed model was evaluated using a mathematical method for performance evaluation of distributed systems [1]. This method is based on the theory of constraints (TOC), with or without manufacturing focus, and on the cost/utilization model. The idea underlying the method is to generalize the application of TOC combined with cost/utilization for performance analysis of a single processor, into a scenario of a distributed network composed of several processors. The method exploits a simple graphic display of the processing element (PE) components (e.g., CPU, Input/Output, Memory, Communication links) in order to pinpoint improper imbalances, fluctuations and bottlenecks. The model uses the following two main indicators for evaluating the performance of a distributed system. The values of \( F \) and \( B \) are between 0 and 1.

\[
F = \text{Cost/utilization Factor} = \sum_{i=1}^{j} P_i * U_i
\]

\[
B = \text{Balance Factor} = 2 * \sqrt{\sum_{i=1}^{j} [(F-U_i)^2 * P_i]}
\]

Where
- \( I \) = Number of processing elements on a single processor
- \( P_i \) = Relative cost of PE \( i \)
- \( U_i \) = Utilization percentage level of PE \( i \)

The closer \( F \) gets to 1 the better the utilization of the network is in terms of the cost of its elements. The closer \( B \) gets to 1 the less balanced the network is and the bigger the variance in the utilization of its elements. Since the percentage of resource utilization in the original model is replaced by the maximal resource utilization in the PE, it is better to have a system that is balanced (a smaller \( B \) is better). If there is a resource that is highly utilized in one of the PEs compared to the other resources in that PE, a moderate increase in the workload might cause a crash or bottleneck at that PE.

The evaluation of the model was performed by comparing the \( B \) and \( F \) measures in two scenarios: hot standby, where a computer is used as a mirror backup (without routinely sharing the workload of the other computers); and a scenario, where the backup computer processes tasks and the load is balanced among all computers linked to the network.

Table 1 depicts the values calculated for \( B \) and \( F \) in the two scenarios. In both cases the utilization of the two computers is not good. The cost of purchasing the backup computer is an imposed operational constraint, and therefore there is no option to alter the cost of the combined system. The reconfiguration model seems to be the preferred option because the system is more balanced (0.433<0.653) and can therefore handle peak processing volume with a better quality of service. In other words, the model enables avoiding bottlenecks which cause down time and impair service to the end-user. In the hot standby option, the risk of a total malfunction, however, is higher because the operations relies only on a single computer which is more prone to crash.

### Table 1: Comparison of Cost/Utilization and Balance Factors

<table>
<thead>
<tr>
<th></th>
<th>Hot Standby</th>
<th>Reconfiguration Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B )</td>
<td>0.365</td>
<td>0.21</td>
</tr>
<tr>
<td>( F )</td>
<td>0.653</td>
<td>0.433</td>
</tr>
</tbody>
</table>

4. Discussion

This study proposed and evaluated an algorithmic model for combining hot standby and load balancing in a network of computers where tasks are processed concurrently and re-allocated by servers running concurrently on different computers.

The research found support for the claim that a combination of fault tolerance and load balancing mechanisms is more effective than software-based fault tolerance alone. The combined approach is also better than implementing a purely hardware-based fault tolerant system, which is a much more expensive solution because it requires the purchase of specialized, synchronized, fault-tolerant computers.

A major advantage of the model is its flexibility and scalability. The model can operate on various hardware platforms and has a great effect on both real-time and data processing applications.

The model can be expanded in the future to include an internal feedback system that changes the vote (relative importance) of different servers automatically to achieve an optimal balance in the network. Such a system will invoke a quantitative model, suggest a modification to the human administrator, and enable “what-if” analysis regarding the effects caused by various changes in the logical configuration of the network.

Another way to evaluate the contribution of the model is by relating it to the work on the GATOSTAR system [4]. The conclusion from this comparison is that the main theme of the GATOSTAR system is the specification of a way to achieve efficient integration of load sharing and fault
tolerant systems, GATOSTAR focuses on a detailed design specification of its ring architecture, and on measuring and comparing the efficiency of the system with previous systems in an academic lab setting.

The main theme of the reconfiguration model presented in this article, on the other hand, is the application of the Hamilton method [5] to the task redistribution process. This article also analyses the effectiveness of the proposed method in a very large-scale industrial setting. A combination of the two approaches is recommended for covering all aspects of the dependability challenge.

References:


