Fault tolerance essay sample

In this day and age, computers have become an integral part of everyday life. Apart from the computers we interact with on a daily basis, there are also numerous computer systems running critical services for our daily lives. Complex computer systems process multitudes of dynamic information handled by institutions like corporations, brokerage firms, telecommunications companies and retail stores. Computer systems are also used to hold and retrieve tons of records for places like insurance companies, email providers and banks. Lastly, computer systems may also operate key systems like traffic control or power distribution networks. While reliability is always a major concern for each system installation, there is a greater need for reliability for these unseen computer systems. Loss of service can mean huge amounts of customer dissatisfaction due to the large numbers of customers which these systems serve.

Fault tolerance is a design strategy which aims to minimize the amount of system downtime. The aim for fault tolerant design is to ensure that the system as a whole is able to give a reliable quality of service even if one subcomponent encounters a fault. Failures should be invisible to the service customers of a computer system. Fault tolerance is applied in many levels of a computer system from fault tolerant power system design (for guarding against loss of power) to database shadowing to security measures put in place to prevent malicious access to the system.

One aspect of fault tolerant design would be system design. In system design, the choice of computer hardware as well as its configuration are geared towards providing a robust and reliable, fault-tolerant system. The primary method of providing fault-tolerance is through redundancy. There would be multiply units of installed critical hardware such that if one unit fails, the redundant unit can carry the load while the broken unit is repaired or replaced.

There are three main redundancy schemes that system planners may choose from with each scheme giving varying levels of redundancy. In One-for-One redundancy, each hardware component has a counterpart backup component. The hardware component in use is called the active unit and the backup is called the standby. The standby will be synchronized with the active unit and will take over operations if the active unit fails. One-to-One redundancy provides the highest level of redundancy as the combined probability of both the active unit and the standby unit failing at the same time is quite low.

Another scheme is the N+X scheme. In this scheme, for N active units, there will be X standby units. The number of standby units is usually less than the number of active units. This provides a lower level of redundancy than one-for-one redundancy but is also lower in cost due to the smaller amount of units to be installed. A higher level component is in charge of monitoring all units and will decide which of the X units to be activated in case one of the N active units will fail.

Lastly, load balancing is a scheme that utilizes no redundant units. In load balancing, a high level module distributes the workload among all active units. If a unit were to fail, the high level module is in charge of redistributing the computing workload among all other functioning units. This is by far the cheapest method of redundancy. However, if a fault occurs during the busiest hours of operation, there would be a significant drop in service quality provided by the system.

In all these redundancy schemes, an important component aside from the active, standby and higher level units is the synchronization between the active and standby. Synchronization is important so that the standby can quickly take over the active unit in case of failure. Just like redundancy, synchronization between hardware units can be achieved in different ways. Each synchronization method differs from one another in the choice of hardware components or events to mimic between an active and standby unit.

In Bus Cycle Level synchronization, the active and standby units are synchronized at the processor bus level. Each instruction executed by the active unit is performed by the standby unit. The standby then compares the output of the two instructions and if the results do not match, it may take over from the active unit. Bus level synchronization is costly due to the need to implement specialized hardware. There is also degradation in system performance due to the need to introduce wait and compare states between cycles. However, bus level synchronization offers the highest level of synchronization between two machines since they are synchronized at the deepest hardware level.

An alternative to bus level synchronization is memory mirroring. In memory mirroring, only one processor is working but the memory of both the active and standby are continuously updated. The single active CPU writes to both memory banks and the output of both memory banks are compared during every read cycle. In case of a mismatch, a parity check is performed and the machine which fails the parity check is assumed to be at fault. The standby processor will take over at this point. Since the memory is synchronized between the two machines, the standby processor can immediately take over program execution. Memory mirroring also needs additional hardware and will degrade performance by introducing wait states.

Further up, the next level of synchronization is message level synchronization. In this scheme, the standby unit receives all external stimuli received by the active unit. The standby will process all inputs but does not transmit its output. This scheme does not need specialized hardware. However, there is a risk of loss of synchronization in complex environments, especially in cases where the active and standby units may make different decisions based on the same external input.

Lastly, the integrator may simply choose not to have any form of active synchronization. In the Reconciliation on takeover scheme, the standby requests all information needed to be in step with the active when it starts to takeover. This scheme is the simplest to implement and will have no overhead performance costs due to the absence of synchronization, However, it is impractical in situations where there may be significant delays in having the standby reconcile with the active unit.

We see that in aiming for reliability through redundancy, the designer of a system has many options regarding the levels of redundancy to be installed in the system as well as the level of synchronization between units. From a fault-tolerance perspective, one will always want the highest levels of redundancy and synchronization. However, these options are the most costly. At the highest level of redundancy, hardware costs will more than double due to the need to purchase two units of every critical hardware component as well as the purchase of the auxiliary hardware needed to keep the two systems synchronized. Many large name corporations manufacture fault tolerant systems. Companies such as HP, Stratus and NEC produce fault tolerant servers which incorporate the highest levels of synchronization and redundancy. While these machines used to cost in the millions of dollars, their cost have come down significantly in recent years so that smaller IT departments are now able to purchase these units for their own use. Low end Stratus ftServer 3000 systems start at $30, 000 and the costs keep on dropping every year.

One should keep in mind that hardware is not the end all of fault tolerant design. Only 10% of the cost of fault-tolerant design goes to the computing hardware. There is also the task of fault-proofing the other components of a computing system such as the power, software, data and other aspects which may fail and degrade performance. More importantly, the cost of these systems should always be compared against the potential costs of downtime.

Bibliography

Avizienis, A.. (n. d.). Dependability And Its Threats: A Taxonomy. In Newcastle University. Retrieved June 7, 2008, from http://rodin. cs. ncl. ac. uk/Publications/avizienis. pdf.

Event Helix. (2007). Hardware Fault Tolerance. In Event Helix. Retrieved June 7, 2008, from http://www. eventhelix. com/RealtimeMantra/HardwareFaultTolerance. htm.

Mears, J.. (January 5, 2004). The new face of fault-tolerant servers. In Network World. Retrieved June 7, 2008, from http://www. networkworld. com/news/2004/0105fault. html.