Reliable Distributed Computing for Decision Support Systems

Taha Osman, Andrzej Bargiela
Department of Computing, The Nottingham Trent University

Introduction

It is widely accepted that the realisation of complex monitoring and control systems, in general, and industrial decision support systems, in particular, is best accomplished using distributed computing resources.

Real-time decision support typically requires that several tasks such as on-line monitoring, simulation, prediction, state estimation, and real-time control are run concurrently. This is implemented most effectively in a true distributed system rather than in a time-shared uni-processor environment. Moreover, the physical size of many decision-support systems such as water distribution networks, involving hundreds or even thousands of pipes and network nodes spread over a large geographical area, calls for a control of computational complexity of the constituent decision support tasks. This is usually achieved by topological decomposition of systems and parallelisation of the respective algorithms[1,8]. The parallel tasks can then be executed on networked workstations which provide a powerful and flexible (upgradable according to specific requirements) computing environment. The performance of such systems has been recently significantly enhanced by the use of high-speed communications links such as FDDI, ATM, Gigabit Ethernet etc [2].

However the use of distributed computing in real-time decision support faces a considerable challenge of providing a reliable service. The ‘hot-standby’ technique of enhancing the fault tolerance of uni-processor systems is quite inappropriate (expensive) when dealing with distributed computing systems. Moreover, the cumulative effect of individual computing node failures makes any distributed system inherently less robust than a uni-processor equivalent.

Consequently it appears that the prerequisite of any distributed computer based Decision Support System is the availability of a software layer that can isolate applications from the hardware faults and ensure that the failure of individual computing nodes does not result in total system failure but only in some degradation of its performance.

The fault-tolerant distributed computing environment (FADI)[3,6], outlined in this paper, was developed at the Nottingham Trent University in order to harness the computational power of interconnected workstations and to provide a fault-tolerant parallel computing platform that meets the practical requirements of advanced decision support systems. FADI supports the automatic (user-transparent) detection of faulty hardware, recovery of the tasks that have been interrupted by the hardware failure, and the re-distribution of application tasks on the currently available computing nodes.

Distributed Systems Reliability

Due to the interdependency of their components, distributed systems are particularly vulnerable to hardware faults. If appropriate fault management measures are not taken, a failure of a single computing node could cause the crash of the whole distributed computing system and consequently the failure of the decision support system it is servicing.

Addressing the reliability issues of distributed systems involves tackling two problems: error detection and process recovery. Error detection is concerned with permanent (such as computer crashes caused by power failure) and transient (such as temporary memory faults caused by electro-magnetic interference) computer hardware faults as well as faults in the software modules and the communication links[3].

Fault-tolerance methods for distributed systems have developed in two streams: checkpointing/rollback recovery and process-replication mechanisms. With process replication techniques, each process is copied and executed on several computers so that the probability that all replicas would fail is acceptably small[4]. A distributed computation should proceed correctly as long as there exists one living replica of each process. These techniques incur a smaller degradation in performance when compared to checkpointing mechanisms, but the main hindrance to their wide adoption in industry is the cost of the redundant hardware that is needed for the execution of the replicas.

In contrast with process replication mechanisms, checkpointing/rollback techniques do not require the duplication of the underlying hardware or the replication of the application processes. Instead, each process periodically records its current state and/or some history of the system in stable storage, an action called checkpointing[5].

The checkpoint contains a complete execution state of a process (e.g state of CPU registers, interrupt handlers, data and stack segments, etc.).

If a failure occurs, processes return to the previous checkpoints (rollback) and resume their execution from the respective checkpoints. The overhead that this technique incurs is greater than that of process replication mechanisms because checkpoints are taken during failure-free operation. Nevertheless, this overhead is getting smaller as the computers and communication networks become more efficient. Hence, the degradation in performance caused by the checkpointing mechanism is quite acceptable for on-line decision support systems.

Fault-Tolerant Distributed Computing Environment

A structural diagram showing the functional decomposition of the fault-tolerant environment (FADI) is illustrated in Figure 1. When execution starts, the Process Allocation module identifies the network configuration and passes the list of active computer nodes (hosts table) to the user and the component FADI processes. Upon receipt of the specifications of the application tasks, through the user-interface, the Process Allocation module spawns the tasks and broadcasts the information about the task-processor mapping to the rest of FADI modules. When the Checkpointing Coordinator receives a list of spawned tasks, it schedulles their checkpointing at the end of every checkpointing interval and saves the checkpoints in a stable storage.

When one of the distributed computing nodes fails the Monitor Host State module detects this fact and informs the Recover Failed Tasks module about the crashed host_ID (host name) so that it won't try to restart (rollback) failed tasks on it.

The failed host_ID is also sent to the monitor user tasks module to determine the IDs of the user-tasks that were running on the faulty host before it has crashed.

These IDs are then sent to the Recover Failed Tasks module so that it can restart the failed tasks from their most recent checkpoint. The failed task IDs are also passed to the Checkpointing Coordinator to suspend the initialisation of their checkpointing.

The Monitor User Tasks module detects user tasks that have exited prematurely due to a transient hardware failure and sends the failed task IDs to the Recover Failed Tasks module. Task IDs of the successfully recovered tasks are broadcast by the Recover Failed Tasks module, so that the Monitor User Tasks module and the Checkpointing Coordinator can resume their monitoring and checkpointing respectively. It is crucial that the Monitor User Tasks module gets the IDs of the recovered tasks because they are being restarted as new executables with different task IDs.

To minimize the failure-free overhead of the checkpointing, a novel non-blocking checkpointing technique has been developed[6]. The technique involves making a copy of the program's data space and using an asynchronous thread of control to perform the checkpointing routines (reading the process state and recording it to disk) while the user process continues the execution of the program code.

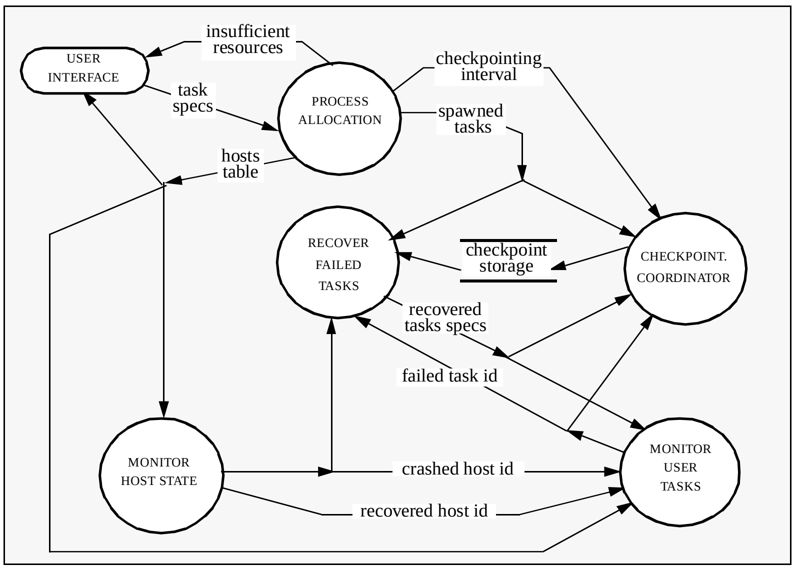

FIgure 1 FADI Structural Diagram

Distributed Water System DSS in FADI Environment

While the FADI enviromnent is largely application independent, it has been developed and tested in the context of a Decision Support System for Water Distribution Networks. A prototype of such a DSS system has been developed, over 15 years ago, by one of the authors and has been ported to a distributed computing environment. The original, single-processor shared-memory implementation of the DSS software[7,9] was re-cast for a cluster of networked UNIX workstations[11]. No changes were made to the algorithmic computing modules written in FORTRAN (the FADI application programming interface supports both FORTRAN and C/C++ applications), but the interprocess communication and synchronisation was restructured to adjust to the facilities of the new environment. The PVM (Parallel Virtual Machine[10]) message passing software was used to facilitate interprocess communication and synchronisation. A centralised control scheme was adopted. The central control task holds the common areas and grants access rights through a request-acknowledgment message exchange with the requesting task schedulled on a FIFO basis.

The principal task of the DSS system is to process redundant, noise-corrupted telemeasurements so as to elliminate the 'bad-data' and to supply a real-time data base with reliable estimates of the current state and structure of the network. The system consists of a number of concurrent software modules, including network simulator, telemetry system simulator, state estimator(s), optimal valve control and the operator interface.

The measurements are processed on a continuous basis (1min scan rate) and the state estimation module identifies discrepancies between the mathematical model of the network and the actual meter readings. These discrepancies are then analysed so that the underlying causes, such as the presence of leakages, closed valves, or erroneous transducer data, are found and remedied[8].

There are three main groups of modules in the package (Figure 2). The modules of the first group simulate the behaviour of the real network and provide measurement information which in real life is acquired through a telemetry system. This data is effectively the only source of information for the second group of modules monitoring the network.

.

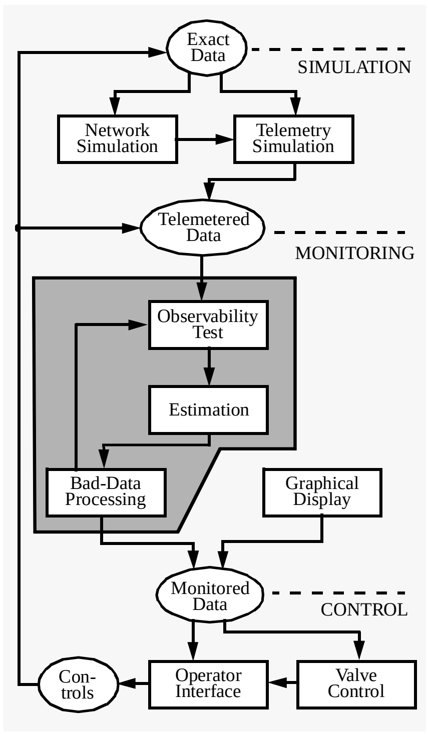

Figure 2 Water Network Monitoring and Control software structure

A major role of the monitoring modules is to validate the information about the system state before it is used by the human operator and the control algorithms. Since the telemetered data is being updated without the intervention of a human intermediary the monitoring modules are said to be on-line to the process.

After checking topological observability of the system, with respect to the current set of valid measurements, the estimates of the state vector are calculated. This is followed by the identification of bad data points which were not found during the preprocessing stage. Depending on the state estimation algorithm employed, the monitoring procedure involves either an iterative elimination of bad data from the set of valid measurements and re-computation of the state vector, or it simply marks erroneous measurements having rejected them in the course of the estimation. The results obtained with the monitoring modules are made available to the operator in the form of a print-out, graphical display and data file which is also used by the control algorithms.

The third group of modules closes the control loop by devising and implementing control actions. The flow of information between the modules implies that algorithmically calculated controls are off-line to the process since they are implemented by a human operator. Such a structure is natural at the initial stage of the computerised monitoring and control of a water network. However, it must be emphasized that the computer assisted control can be easily converted into a full on-line control scheme since the system is monitored on-line.

The water networks monitoring and control application is inherently distributed. Central synchronisation and control, simulation, telemetry, estimation, and operator interface tasks need to run concurrently to keep the distributed control system on-line. The 'control-loop' structure of the DSS software implies that, the failure of any of the computing nodes executing individual software modules will affect the entire system. For example, if the node processing the state estimation crashes, the state estimates will rapidly run out-of-date and gradually become irrelevant to the real-time valve control module. Hence the water networks monitoring and control application has fundamental requirements for both distribution and reliability that only fault-tolerant distributed computing environments such as FADI can deliver.

The Evaluation

The FADI-based implementation of the Water System DSS has been tested, using several Sun SPARC workstations (two SPARC_IPC, one SPARC_10 and one SPARC_20), for a medium-size (92-nodes) water distribution network. PVM-TCP/IP was used for communication over a 10 Mbit/sec Ethernet.

The computational load associated with the execution of the constituent DSS modules strongly depends on the variation of the meter readings, that are being processed, and the presence of measurement errors that need to be filtered-out. For a typical pattern of water system operation, a sequence of five monitoring cycles was accomplished in under 2 minutes with an average CPU load of approximately 30% and a message exchange rate between the distributed applications of 42 messages per second. This illustrates that the Water System DSS software represents a significant challenge from the computation and communication load point of view.

In this context the efficiency of the novel non-blocking checkpointing introduced into FADI has been evaluated. Figure 3 shows the relationshop between the checkpointing interval and the application failure-free overhead.

The failure-free overhead at checkpointing intervals of half a minute approaches 10% of the application running time, but for the default measurement scan rate it is below 5%. The diagram provides a basis for ballancing the concerns of fault-tollerance and the efficiency as required by various applications. The advantage of FADI is that it enables the customisation of the computing environment.

.

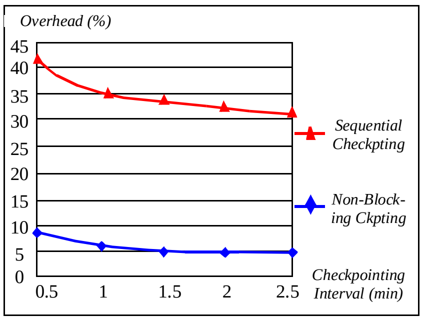

Figure 3 Overhead of the fault-tolerant execution of the DSS system

The ability to recover from the distributed system failure was tested by powering-down or disconnecting from the Ethernet the individual workstations running the computational modules. The FADI environment performed the identification of the hardware faults and the re-allocation of the affected computational modules to the operational computing nodes in a totally user-transparent way. The average recovery time for the water distribution networks application was 8 seconds.

Conclusions

The fault-tolerant environment (FADI) enables reliable execution of concurrent, distributed applications despite hailures of the underlying computing hardware. This integrated environment encompasses all aspects of modern fault-tolerant distributed computing: automatic remote process allocation, detection of hardware errors, and a technique to recover distributed user processes running under FADI from these errors.

Water distribution networks DSS exemplifies a class of industrial systems where the FADI environment can be usefully deployed. The software system is inherently distributed and it needs to execute on a continuous basis. It can tolerate a small delay in the operation of one or more of its tasks (while they are rolled back and restarted from checkpoints backup), but a complete halt of the system can lead to a critical failure of the decision-support system. FADI system, developed at the Nottingham Trent University, represents a low-cost and efficient alternative to hardware-redundancy based systems.

The experimental results confirm that, due to the non-blocking checkpointing methodology, FADI imposes low overhead on the running time of applications. The efficient failure recovery under FADI is bound to be a major asset in the context of industrial system implementations.

References

1. A. Bargiela, A. Argile, and J. Hartley. "Parallel Processing for Probabilistic Decision Support in Water Distribution Systems", SERC Seminar, Brunel University, September 1993.
2. M. Hurwicz, "Preparing for the Gigabit Ethernet. Byte Special Report on Extending the Enterprise", Byte, Vol. 22 N0. 10, p. 63, October 1997.
3. Taha Osman, and Andrzej Bargiela. "Error Detection For reliable Distributed Simulations", In proceedings of the 7th European Simulation Symposium, p. 385-362, 1995.
4. B. Appel et al. "Implications of Fault Management and Replica Determinism on the Real-Time Execution Scheme of VOTRICS".
5. L. Silva and G. Silva. "Global Checkpointing for Distributed Programs", Proc. of the 11th Symposium on Reliable Distributed Systems, Huston, Texas, p. 155-162, Oct. 1992.
6. T. Osman and A. Bargiela, "Process Checkpointing in an Open Distributed Environment", In the Proc. of the 11th European Simulation Multi-Conference, p. 536-541, Turkey, June 1997.
7. A. Bargiela. "On-Line Monitoring of Water Distribution Networks", PhD Thesis, Faculty of Science, University of Durham, May 1984.
8. A. Bargiela and J. Hartley, "Parallel Simulation of Large Scale Water Distribution Systems", Proceedings of the 9th European Simulation Multi-Conference, 1995.
9. A. Bargiela and D. Al-Dabass. "A Simulated Real-Time Environment for Verification of Advanced Water Network Control Algorithms", Systems Science Journal, Vol. 14, No. 3, 1988.
10. A. Geist et al. "PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing". The MIT Press, Cambridge, Massachusetts, 1994.
11. M.C.Chen "Distributed processor implementation of a water system monitoring and control", Research Report, NTU, 1996