Volume 2, No. 3, March 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

Fault Tolerance by Replication in Parallel Systems

Madhavi Vaidya
Affiliated to University of Mumbai,
India
vamadhavi03@yahoo.co.in

Abstract: In this paper the author has concentrated on architecture of a cluster computer and the working of them in context with parallel paradigms. Author has a keen interest on guaranteeing the working of a node efficiently and the data on it should be available at any time to run the task in parallel. The applications while running may face resource faults during execution. The application must dynamically do something to prepare for, and recover from, the expected failure. Typically, checkpointing is used to minimize the loss of computation. Checkpointing is a strategy purely local, but can be very costly. Most checkpointing techniques, however, require central storage for storing checkpoints. This results in a bottleneck and severely limits the scalability of checkpointing, while also proving to be too expensive for dedicated checkpointing networks and storage systems. The author has suggested the technique of replication implemented on it. Replication has been studied for parallel databases in general. Author has worked on parallel execution of task on a node; if it fails then self protecting feature should be turned on. Self-protecting in this context means that computer clusters should detect and handle failures automatically with the help of replication.

Keyword: Architecture, paradigms, data, dynamically, checkpointing techniques

INTRODUCTION

There is as such no simple recipe for designing parallel algorithms. The design methodology allows the programmer to focus on machine-independent issues such as concurrency in the early stage of design process, and machine-specific aspects of design are delayed until late in the design process. The goal of parallel paradigm is decomposition of the complete task into subtasks that are as independent as possible. Distribution of the subtasks over the various processors minimizes the total execution time. Finally, the whole task gets executed in the faster manner. For clusters, the distribution of the data is done over the nodes minimizing the communication time. A cluster is a group of independent servers that function as a single system.

As suggested by Ian Foster, this methodology organizes the design process into four distinct stages [IF96]:

- Partitioning
- Communication
- Agglomeration
- Mapping

PARTITIONING

It refers to decomposing of the computational activities and the data on which it operates into several small tasks. The decomposition of the data associated with a problem is known as domain/data decomposition.

COMMUNICATION

It focuses on the flow of information and coordination among the tasks that are created during the partitioning stage. The nature of the problem and the decomposition method determine the communication pattern among these cooperative tasks of a parallel program. These tasks are executed on the clusters of workstations (cluster computing), it is being used as they are cost effective and scalable and they fulfill the need of high performance computing.

AGGLOMERATION

If required, tasks are grouped into larger tasks to improve performance or to reduce development costs. Also, individual communications may be bundled into a super communication.

MAPPING

It is concerned with assigning each task to a processor such that it maximizes utilization of system resources (such as CPU) while minimizing the communication costs.

In the world of parallel computing there are several authors who have presented a paradigm classification. Not all of them propose exactly the same one, but we can create a superset of the paradigms detected in parallel applications. [LS & RB]

For instance, in [DP], a theoretical classification of parallel programs is presented and broken into three classes of parallelism:

- processor farms, which are based on replication of independent jobs;
- Geometric decomposition, based on the parallelization of data structures;
- Algorithmic parallelism, which results in the use of data flow.

Another classification was presented in [PBH]. The author studied several parallel applications and identified the following set of paradigms:

- pipelining and ring-based applications;
- divide and conquer;
- master/slave; and
- Cellular automata applications, which are based on data parallelism.
-

To summarize, the following paradigms are popularly used in parallel programming: [LS & RB]

- Task-Farming (or Master/Slave)
- Single Program Multiple Data (SPMD)
- Data Pipelining
- Divide and Conquer

TASK-FARMING (OR MASTER/SLAVE)

The task-farming paradigm consists of two entities: master and multiple slaves. See **Fig.1** The master is responsible for decomposing the problem into small tasks (and distributes these tasks among a farm of slave processes), as well as for gathering the partial results in order to produce the final result of the computation. Master process acts as a co-ordinator. One **master process** supervises the execution of the program. It defines independent **tasks** and puts them in a list which is observed by each cluster executing the tasks. It also **collects** the **results** of these tasks.

The slave processes execute in a very simple cycle: get a message with the task, process the task, and send the result to the master. Any number of **slave processes** each takes a task from the list, execute it, and put the result into the master's mailbox.

Usually, the communication takes place only between the master and the slaves.

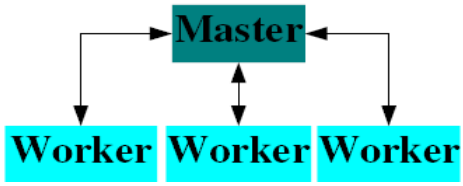

Figure.1

Advantages:

- Very simple model
- No deadlocks, since only the master process ever waits for another process to finish

Limitations:

- Tasks cannot delegate work to sub-tasks. Adding this possibility would introduce deadlocks.
- Rigid communication pattern, no optimization possible.
- Distributed data storage impossible.

Task-farming may either use static load-balancing or dynamic load-balancing. When the distribution of tasks is all performed at the beginning of the computation, which allows the master to participate in the computation after each slave has been allocated a fraction of the work. Refer **Fig.2.** The allocation of tasks can be done once or in a cyclic manner; in this way the static load balancing works.

A dynamically load-balanced master/slave paradigm, in which the number of tasks exceeds the number of available processors, or when the number of tasks is unknown at the start of the application, or when the execution times are not predictable, An important feature of dynamic load-balancing is the ability of the application to adapt itself to changing conditions of the system.

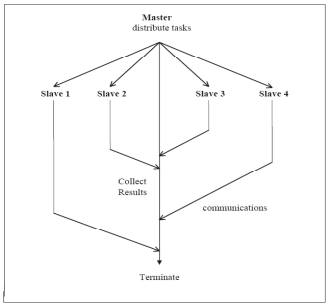

Figure.2

Due to this characteristic, this paradigm can respond quite well to the failure of some processors, which simplifies the creation of robust applications that are capable of surviving the loss of slaves or even the master. But the drawback is serialization (process running should not clash with each other) from master to the slave tasks. Processing and network communication among them cause an important hidden cost in this technique. The tasks list and result list occupy memory in the master process.

SPMD(Single Processor Multiple Data)

Each process executes basically the same piece of code but on a different part of the data. This involves the splitting of application data among the available processors.

SPMD applications can be very efficient if the data is well distributed by the processes and the system is homogeneous. If the processes present different work-loads or capabilities, then the paradigm requires the support of some load-

balancing scheme able to adapt the data distribution layout during run-time execution.

DIVIDE AND CONQUER

The divide and conquer approach is well known in sequential algorithm development. A problem is divided up into two or more subproblems. Each of these subproblems is solved independently and their results are combined to give the final result. In parallel divide and conquer, the subproblems can be solved at the same time, given sufficient parallelism. The splitting and recombining process also makes use of some parallelism, but these operations require some process communication. However, because the subproblems are independent, no communication is necessary between processes working on different subproblems.

Data Pipelining

Processes are organized in a pipeline; each process corresponds to a stage of the pipeline and is responsible for a particular task. The communication pattern can be very simple since the data flows between the adjacent stages of the pipeline.

Parallel System's - Node Breakdown

Increase in the number of components in such systems increases the failure probability. To provide fault tolerance it is essential to understand the nature of the faults that occur in these systems. There are mainly two kinds of faults: permanent and transient.

Permanent faults are caused by permanent damage to one or more components and transient faults are caused by changes in environmental conditions. Permanent faults can be rectified by repair or replacement of components.

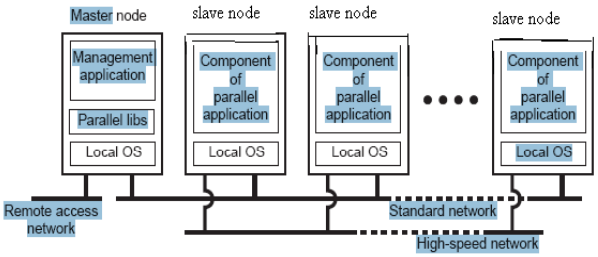

Diagram for Task Farming (Master/Slave)
Figure.3

The concept of a failure refers to a single node failing in an actively processing application, other than a software error. If there is a software error, then the bugs appearing in it can only be repaired by recoding the application. The failures can be defined as a breakdown in communications between nodes. Observe **Fig.3** To this end, to detect failures, all that needs to be done, is for multiple machines to monitor each other's conditions.

Tasks may get finished for many reasons other than communication breakdowns, that aren't necessarily software faults - such as running out of memory. It may be a node failure problem. If a node fails, you could just repeat the work elsewhere. If the process is been running for a longer time this may result in a significant amount of lost work. In this scenario, what can be done is a method can be applied of saving the state of programs at certain intervals, so that in the event of failure we could start from a more recent position and minimize the loss of work. Here, the concept of checkpointing is used.

CHECKPOINTING

Checkpointing can be done at various levels. Ex. Process failing, logging all messages, node failure. One way of checkpointing is to log all of the messages and interaction between tasks. If a process fails, and needs to be resumed, then resend the messages that it received from the log to get it to the point in the program where it was before. If messages are sent frequently, then this approach may be sufficient and not lose much work. Another approach is to use process level checkpointing, this type of checkpointing works very well for single processes.

With a parallel system, the failure of one node may be unrecoverable many a times, or at the very least it will take and unacceptable length of time for the original machine to resume. This leads to the requirement of process migration as explained by the author[KO02]. A failed process (for whatever reason) may need to be resumed on an alternative node than the one it was previously working on. Assuming all nodes are effectively similar, that is they are running on homogeneous system, the checkpointing data of each node needs to be available to the entire pool of processors. To make available the checkpointing data , the data can be stored either on each node (requiring huge duplication and a heavy load on the network) or stored at a central location , we can say if task farming is used then at the master's site.
The ability to migrate a process may be exceptionally useful in this situation, as the process in hand can simply be moved elsewhere. This principle can be extended to the task of load balancing.
All the discussion we have done till now was with the situation when the homogeneous nodes are in use. But what if, the number of nodes used is heterogeneous.

HETEROGENEOUS CLUSTERS

Heterogeneous workstations clustered together, where each workstation may have CPUs with different performance capabilities and differing amounts of memory and caches, and even different architectures and operating systems. There is another type of heterogeneity. Even if all workstations are identical, with identical CPUs, memory and caches, they may not be 100% dedicated to the task for which they are being used. Any or all of these processors may be executing other applications that reduce the performance capacity of the processor that is available for a particular application.

In addition the load on these processors may vary dynamically over time as it is dependent on applications running on the machine, which means the capacity dedicated to a particular task may vary.

CHECKPOINTING ON HETEROGENEOUS CLUSTERS

In the said paper, [KW] these two authors have discussed the method of checkpointing on heterogeneous nodes as ,the goal of their checkpoint propagation approach is to generate a variety of specific machine-dependent checkpoints for heterogeneous systems. One straightforward way they suggested would be to replicate processes on each type of machine in the network. PREACHES, a checkpointing tool for single process applications in heterogeneous systems, has been successfully developed by them using the checkpoint propagation technique. They used the term "receiver makes right" mechanism is implemented to reduce the overhead of data packing and unpacking. In short, these authors have tried to concentrate on the fault detection and recovery facility which can be done with the support of portable checkpointing and process migration in heterogeneous systems.

DISADVANTAGES OF CHECKPOINTING

A major issue of checkpointing is the degradation in performance. Taking a checkpoint may require storing an awful lot of data - this can take some time.Moreover, on a parallel machine, a checkpoint is taken at the same stage over the whole network; this often requires all processes to be synchronized before the checkpoint.

If the number of processes get executed from heterogeneous machines then it is more difficult. It may create degradation on the performance of the clusters running in network.

If a node fails, the system cannot restart by remapping checkpoints to existing nodes. Instead, a new node must be inserted into the cluster to force the restart topology into consistency with the original topology. This requires the existence of spare nodes that can be allocated to the computation to replace failed nodes.

Instead of checkpointing if the replication technique is used in the scenario of a node failure then it helps to avoid the faults generated in the system. It is explained in the below listed benefits.

MY IDEA OF IMPLEMENTATION

The author has suggested the replication as a methodology which can be used in the place of checkpointing. The replication of data in a parallel distributed system offers the benefits like the data can be copied on all clusters if needed or the master copy is kept at Master node of all clusters and then can be replicated or used wherever needed. This will create a lot of duplication hence the replication technique by using secondary failure is been explained here.

REPLICATION

Replication is used as a solution to fault generating systems. Failure resilience is also difficult to achieve. If a site fails, the data it contains becomes unavailable. By keeping several copies of the data at different sites, single site failures should not affect the overall availability which is the aim of primary copy approach discussed under replication.
There are two types of replication techniques: **Synchronous and Asynchronous**

Synchronous Replication: Synchronous replication propagates any changes to the data immediately to all existing copies.
Asynchronous Replication: Asynchronous replication first executes the updating operation on the local copy. Then the changes are propagated to all other copies. While the propagation takes place, the copies are inconsistent

By implementing either of them, the author thinks that it will be copies of the data maintained at each site and hence will increase the wastage of the memory by keeping copies of the similar data on many sites.

ADVANTAGES OF REPLICATION

Increased availability

One of the important advantages of replication is that it tolerates failures in the network gracefully. The system remains operational and available to the users despite failures. By replicating critical data on servers with independent failure modes, the probability that one copy of the data will be accessible increases. Therefore alternate copies of a replicated data can be used when the primary copy is unavailable.

Increased reliability

Many applications require extremely high reliability of their data stored in files. Replication is very advantageous for such applications because it allows the existence of multiple copies of their files. Due to the presence of redundant information in the system, recovery from catastrophic failures becomes possible.

Improved response time

Replication also helps in improving response time because it enables data to be accessed either locally or from a node to which access time is lower than the primary copy access time. The access time differential may arise either because of network topology or because of uneven loading of nodes.

Reduced network traffic

If a file's replica is available with a file server that resides on a client's node, the client's access requests can be serviced locally, resulting in reduced network traffic.

Improved system throughput

Replication also enables several client's requests for access to the same file to be serviced in parallel by different servers, resulting in improved system throughput.

Better scalability

As the number of users of a shared file grows, having all access requests for the file serviced by a single file server can result in poor performance due to overloading of the file server. By replicating the file on multiple servers, the same requests can now be serviced more efficiently by multiple servers due to workload distributed. This results in better scalability.

Autonomus operation

In a distributed parallel system that provides file replication as a service to their clients, all files required a client for operation during a limited time period may be replicated on the file server residing at the client's node. This will facilitate temporary autonomous operation of client machines. A distributed system having this feature can support detachable, portable machines.

- The load at the primary copy can be quite large
- Reading the local copy may not yield the most up to date value

In the parallel and distributed systems research area replication is mainly used to provide fault tolerance. Many papers have discussed about the replication strategy used in parallel as well as distributed databases. In the parallel computation paradigm, the main tasks (process) when executed on many nodes in a parallel manner, two replication strategies have been used in distributed systems: **Active** and **Passive** replication.

In Active Replication each client request is processed by all the servers. All processes will be given the same initial state and a request sequence, all processes will produce the same response sequence and end up in the same final state. Either all the nodes will receive a message or none, plus that they all receive messages in the same order

In Passive replication there is only one server (called primary) that processes client requests. After processing a request, the primary server updates the state on the other (backup) servers and sends back the response to the client. If the primary server fails, one of the backup servers takes its place.

The author has described the primary and secondary failure techniques. The secondary failure technique resembles with the passive replication where data is not copied simultaneously on many machines. But if the failure of a node occurs then the fault tolerance technique through self protection mechanism of replication gets started.

Replication is used for performance and fault tolerant purposes. By replication, the recovery procedure is executed for failed replicas. [18P] Node failure handling is done by using two techniques:-

- **Secondary failure –** this technique uses the function as nothing to do till the (failed) node recovers
  - ✓ At recovery, apply the updates it missed while the cluster was down
  - ✓ Needs to determine which updates the specific cluster missed, but this is no different than log-based recovery
  - ✓ If the node is down for too long, may be faster to get a whole copy
- **Primary failure** – all other processes just wait till it recovers
  - ✓ Can get higher availability by electing a new primary
  - ✓ A secondary that detects primary's failure announces a new election by broadcasting its unique replica identifier
  - ✓ Other secondaries reply with their replica identifier
  - ✓ The largest replica identifier wins

The author suggests that the Secondary Failure Technique should be used for the problem of checkpointing which is discussed in the paper.

Secondary Failure Technique will help the system to overcome the failure problem of a node.
If the Secondary Failure technique elects a new primary and the old primary is still running, there will be a reconciliation problem when they're reunited. This is multi-master. There is a type of settlement done in the original primary node and the newly repaired one. Hence it is known as multi master.

CONCLUSION

Checkpointing is a very useful technique, which is, used parallel computation for fault tolerance. Check pointing is a way of taking a snapshot of tasks at any point of time, so that in the event of failure, the task can be resumed from that point. But it has also some drawbacks which author discussed already. A major issue of checkpointing on a parallel machine is the user has to take a checkpoint at the same stage over the whole network, this often requires all processes to be synchronized before the checkpoint.

Hence author suggests the replication technique by implementing the recovery procedure with the help of two techniques Passive Replication and Secondary Failure. Out of them, author felt the secondary failure technique more useful from this issue's point of view. Reason behind selecting this secondary failure is, if some/one of the cluster node goes down, its updates can be stored by replication's self-protect mechanism. How the log based recovery is done, in the same manner the recovery from the failed node is done and the tasks which would have been executed on that specific node will be accomplished by other nodes available in the link.

REFERENCE

[1] [IF96] I. Foster. Designing and Building Parallel Programs. Addison Wesley, 1996, available at http://www.mcs.anl.gov/dbpp

[2] [KO02] Fault Tolerant Parallel Programming

[3] [LS & RB] Parallel Programming Models and Paradigms Luis Moura Silvay and Rajkumar Buyya

[4] [PBH] P. B. Hansen. Model Programs for Computational Science: A Programming Methodology for Multicomputers. Concurrency: Practice and Experience, vol. 5 (5), pages 407-423, 1993.

[5] [DP] D. Pritchard. Mathematical Models of Distributed Computation. In Proceedings of OUG-7, Parallel Programming on Transputer Based Machines, IOS Press, pages 25-36,

[6] [18P] research.microsoft.com/18_Philbe Replication Standford99.ppt -- 1999 Philip A. Bernstein

[7] [KW] PREACHES, Portable Recovery and Checkpointing in Heterogeneous Systems

[8] Kuo-Feng Ssu W. Kent Fuchs

