

Modeling Work Flow in Hierarchically Clustered Distributed Systems

Austin Gilbert, Johnson Thomas, István Jónyer
Department of Computer Science
Oklahoma State University
219 MS, Stillwater, OK 74078

Abstract

The use of hierarchical clustering in distributed systems shifts the burden of work flow management from a central server to work managers participating in the system, as well as necessitating a node profiling mechanism. Examination of this shift illuminates the need for alternative approaches to managing work unit distribution and result collection. This paper describes our intended initial approach to work assignment along with a simple formal model for work flow in the system and some of the issues that the system will need to address.

Keywords: Work Flow, Distributed Systems, Petri Nets

1. Introduction

Many of interesting problems in science, especially biology, chemistry and biochemistry, remain unresolved due to their great computational complexity. There are times, however, when computational complexity can be overcome by taking advantage of parallel aspects of problems. Distributed systems aim to exploit this by distributing the computation among networked computers. Often they out perform the world's fastest super-computers, but this performance is gained at the cost of bandwidth utilization on the central data server. To reduce the bandwidth demands created by large-scale distributed systems, we are developing a hierarchically clustered distributed system in which we allow the aggregation of clients into logical computation units, or clusters. In every cluster one member node will become responsible for downloading and distributing the work for the other cluster members, thereby reducing the number of clients the central server must keep track of. Further, the system will allow for clusters of clusters in order to maximize the benefits of locality among clients. The primary benefits of clustering include shorter download times for the clients, and reduced traffic on the central server. The primary focus of this paper will be to describe our algorithm for the amount of work assigned to each client when requests are made, and to formally model the data flow among the clusters, clients, and work-managers.

The remainder of this paper is organized as follows. Section 2 describes our approach to node profiling and work assignments, section 3 presents a formal model of our approach, finally, conclusions are presented in section 4.

2. Work Flow Management

In current variations of large-scale distributed systems, such as Seti@Home and Folding@Home, each client node (worker) is assigned one work unit at a time. The clients process the work unit until completion, at which point the results are returned to the central server (or a result-server) for storage and later analysis. The apparent problem with this approach is that faster computers may continually be contacting the central server to request and return work. This pattern of frequent and unscheduled communication with the central server could reduce the communication-to-computation ratio of the overall system and increase the likelihood for traffic congestion, both of which risk reducing the system's overall performance. A second issue with the centralized distributed system approach is that all the work flow (i.e. work assignment and the returning of results) must be handled by a central server (or logical cluster of servers) which can produce large and sporadic demands for bandwidth and also tends toward traffic congestion at times.

The apparent solution to these issues is three-fold. First, assign faster clients more work to keep them busy longer in order to maintain the desired low communication to computation ratio. Second, schedule the work unit downloads and uploads. Finally, delegate the work management from the central server to the clients able to perform such tasks. Although the idea of distributing the work flow management among the clients to form clusters and clusters of clusters is central to our hierarchical clustering model, our primary focus in this paper will be node profiling and work assignment. As such, when we refer to a *management node*, a *work manager*, or a *management server*, for all practical purposes we are discussing any node in the system responsible for distributing/collecting work, be it the central server or a client taking on additional responsibilities as a cluster manager. The approach used to determine download/upload schedules and cluster formation algorithms and their impact on the system performance is considered out of the scope of this paper, and will be developed at a latter date.

2.1 Node Profiling and Work Assignment

Assigning clients with faster processors more work than slower clients introduces the need for node profiling of the clients, i.e. having access to their past performance through a historical record becomes necessary. In [2] we discussed the use of a reputation system as a means of embodying the actions of participating nodes over time in order to facilitate data integrity in the system. Here we will explore the reputation system as a means of embodying the computation performance of a node over time. The simultaneous use of a reputation system to perform widely divergent functions within the system is evidence of its elegance and flexibility.

Every node in the system will have a speed reputation attribute, or a value representing the abstracted combination of bandwidth and CPU performance. The faster a node is able to download, complete, and return a work unit, the higher the node's speed reputation value. Likewise, the longer the duration, the lower the reputation. The reputation of a worker node will then be manipulated by the work-manager the client node (worker) is assigned to, raising the value relative to *good* performance, and lowering it with *bad*

performance. Hence, the speed attribute eventually becomes a representation of the client's performance metric relative to the other nodes participating in the system.

The work manager will then use the speed reputation of clients to determine the amount of work to assign to a worker during a work request. The intention is to assign multiple work units simultaneously. To facilitate this, the work manager will bundle multiple work units together and assign them to a client. Once the client has received the work, if the client is also a work manager (in a cluster of cluster situation) then the client may distribute the work among its workers for processing. As the work units are finished, a client's completed results are returned to their work manager, and their manager then returns the work to their manager, and so on until the root manager (the central server) is reached.

The amount of work to assign during each request is a primary concern. Here we intend to use an algorithm similar to the *binary exponential backoff algorithm* used in TCP. Our algorithm for the number of work units assigned is described as follows:

- Let the optimal time lapse between client/server communications be called T .
- Let P_i be the identification for a logical computational unit (either a client or a cluster).
- Let $W(P_i) = \#$ of work units previously assigned to the client P_i (or equal to 1 if this is the first work assignment for the client P_i).
- Let t be the time lapse between client P_i downloading a work unit and returning it.
- Let $f(P_i)$ be a function describing the work load adjustment to be made by the work manager.
- Let $p(P_i)$ be equal to the value of $f(P_i)$ from the immediately previous interaction.
- Let $f(P_i)$ be defined as such:
 - If $t \ll T$ then $f(P_i)$ is defined as:
 - if $p(P_i) = 0, f(P_i) = 1$
 - if $p(P_i) = 1, f(P_i) = 2$
 - otherwise $f(P_i) = p(P_i)^2$
 - If t near T then $f(P_i) = 0$
 - If $t \gg T$ then $f(P_i)$ is defined as:
 - if $p(P_i) = 0, f(P_i) = -1$
 - if $p(P_i) = 1, f(P_i) = -2$
 - otherwise $f(P_i) = -p(P_i)^2$

- Let $A(P_i) = W(P_i) + f(P_i)$

Then, for each client P_i 's work request, we assign $A(P_i)$ work units to the client to process, and modify P_i 's speed reputation according to their performance by an arbitrary amount. If $f(P_i)$ is positive, then increment the speed reputation of P_i . If $f(P_i)$ is negative, then decrement the speed reputation of P_i .

Therefore, if a worker is able to download, process, and return results for a work unit (task) in less time than the desired optimal time T , then we assign the client additional work. If the client completes and returns their work in an amount of time near the optimal then we assign them the same amount of work that they were assigned previously. If they return the work in an amount of time significantly longer than the optimal T , then less work will be assigned to them. The work increases and decreases are done exponentially. Initially a client is assigned one work unit, if they complete it before the desired threshold T , then they will be assigned 2 work units. If they complete the 2 work units in under time T , then they will be given 4 work units, and so on. The decay works similarly in the opposite direction.

Here we have described our planned algorithm for determining the quantities of work units (tasks) to be assigned to a client/cluster in response to a work request. In the following section a simple formal model for describing the flow of work between managers and clients is presented.

3. Petri Net Model of Work Flow

In the proposed hierarchical clustering system, clients send their completed tasks to the local manager. The manager when it has accumulated a sufficient number of completed tasks forwards the completed task set higher up the hierarchy to its manager. Given the complexity of the system, a formal model is required to verify the correctness of the data pathways. In this section, we model the workflow as a Petri Net. Petri nets are directed graphs [1]. They include Places, Transitions, Token, and Arcs. Places contain tokens. Token could be data or truth of a condition. Transitions are allowed to fire only when they are enabled. In other words, all the input places to a transition have at least one token. A marking specifies the distribution of tokens in the net. In a net the token distribution (marking) is changed on firing the enabled transition. A sequence of firings will result in a sequence of markings. A marking M^1 is said to be reachable from a marking M if there exists a sequence of firings that transforms M to M^1 . A timed Petri net model of a Grid is a tuple $TPN = (P, T, F, TS, D, M)$ where:

- P is a finite set of files, jobs or resources.
- T is a finite set of transitions. Transitions represent tasks that are to be executed by processors.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.
- TS is the time set.

- $D : T \rightarrow TS$ is a function which defines the execution time of each transition.
- $M: P \rightarrow \{0,1,2,3,\dots\}$ is the initial marking,

In particular we model two properties of the grid: liveness and safeness. A transition t in a Petri net TPN with initial marking M is said to be live if and only if for any M' in the set of reachable markings, there exists a marking reachable from M' in which t is enabled. A Petri net is live if all its transitions are live. A place is safe if it is one-bounded ($M(p) \leq 1$) and a Petri net is safe if and only if all its places are safe. Liveness indicates that the tasks in the grid will successfully complete and not deadlock, whereas safeness indicates that unique values to a computation are produced. It is impossible to manually verify the safeness and liveness of a complex system such as the grid.

3.1 Task aggregation

At least one of the managers has to verify that the tasks submitted were successfully completed and the result produced is a unique correct value. Tasks are represented as Petri Nets and these tasks are aggregated and merged to create a Petri Net representation of the entire task. Figure 1 shows a manager and two clients. The manager assigns tasks to the clients and the clients return results to the manager. Figure 2 shows the tasks executed by each client. The results from each client are sent to the manager who may do further processing to generate the final result. In practice, data may be shared between clients. This is not indicated in the figure below. The simple figure below does not capture many of the practical aspects of a real grid. A lack of space in the short-paper format prevented a more detailed explanation and illustration of the Petri net model.

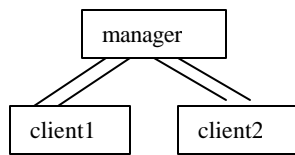


Figure 1: Manager and two clients

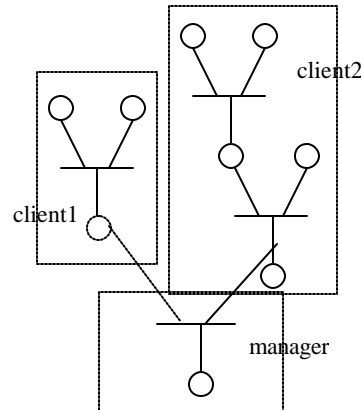


Figure 2: Petri Net model of tasks executed by clients and manager

Given that tasks and sub-tasks are distributed throughout the system, there is great potential for deadlocks and conflicts. We show that if tasks are divided and distributed systematically, the tasks graph will be free from deadlock (live) and will produce unique correct values (safe). There are two steps in generating Petri nets that are live and safe. Firstly, when tasks are assigned by a manager, the tasks have to be partitioned such that the resulting sub-tasks are live and safe. Secondly, as shown in figure 2, when the outputs from the different tasks are integrated and merged at a manager site, it must be shown that the integrated merged task graph is live and safe. It is beyond the scope of this paper to specify the graph partitioning mechanism. We assume that each partitioned task

assigned to each client is live and safe. However, when the outputs are merged and integrated, we get the following:

Lemma 1: The merging of two live and safe Petri Nets does not always result in a net which is also safe and live.

Proof: If a loop is introduced in the merged integrated net, the net may not be safe and live. See figure 3 for example. The merging occurs at the colored places. The resulting integrated net is not live and safe, although the individual nets are.

The types of nets that can be merged are therefore restricted so that the merged net is safe and live. We introduce a constraint on the types of merging that are permitted. We define a merging constraint:

Merging constraint: Any circuit introduced into a net as a result of merging two or more live and safe Petri nets must be a live and safe circuit given an initial marking M_{ini} for the merged net.

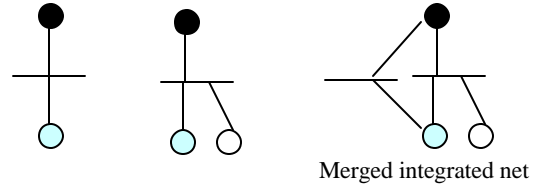


Figure 3: Liveness and safeness of net

Theorem 1: A net TPN formed by merging and integrating multiple safe and live Petri nets $TPN_1, TPN_2, \dots, TPN_i$ is live and safe if the merging constraint is satisfied.

Proof Outline: If no loops are introduced, the net remains live and safe. Any loop will be live and safe due to the merging constraint. Space limitations prevent a detailed proof and explanation of the merging constraint.

In this section we have shown that a formal graph model ensures that tasks assigned by a manager to individual clients will terminate correctly and not cause deadlock. Although the complexity of the grid requires the development of formal models, the role of formal models in the grid has received scant attention. In this section we have proposed a Petri net model for capturing grid work flow such that desirable properties are ensured. Clearly this is only the beginning of such a model and formal models for the division of tasks and resource allocation remains to be done. The above work will also be extended to model the hierarchical grid architecture proposed in this paper.

4. Conclusions

In this paper, we have outlined an initial approach to work unit assignment in a hierarchical clustering distributed system architecture, and provided a simple formal model for describing the flow of data in such a system. Furthermore, we have used the formal model to demonstrate that through the use of constraints on tasks (work units) that the results will be correct once the completed results are assimilated at a central location.

REFERENCES

- [1] Tadao Murata. *Petri nets: Properties, analysis, and applications*. Proceedings of the IEEE, 77(4):541--580, April 1989.
- [2] A. Gilbert, A. Abraham, M. Paprzycki, "A System for Ensuring Data Integrity in Grid Environments," in proceedings of the *International Conference on Information Technology (ITCC'04)*, Las Vegas, Vol. 1, 435-439, 2004.