Towards a Highly Flexible and Highly Scalable Distributed Computing Architecture

Kenan Kalajdzic <kenan@unix.ba>

Written in February 2012 (original idea 2004)

Abstract

This paper presents an idea for building a highly flexible and scalable distributed computing architecture based on the Master-Worker model. The proposed architecture allows for a straightforward implementation of important features such as redundancy, automatic load balancing, network reconfiguration, and offline computation.

1 Introduction

In the traditional Master-Worker architecture, every node in the distributed system has a precisely defined role. There is usually one primary master which distributes the work among one or more of its workers (Figure 1). Some variants add secondary masters to increase fault tolerance and sometimes submasters are used to offload the work from the primary master.

In this paper we propose a different architecture, in which every node in the distributed system has a dual nature. A node can be a master for one or more workers, and at the same time act as a worker of some other master. The concept is illustrated in Figure 2.

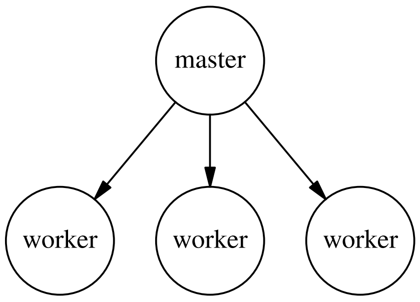

Figure 1: Traditional Master-Worker model. Each node has a single, unique role (either master or worker).

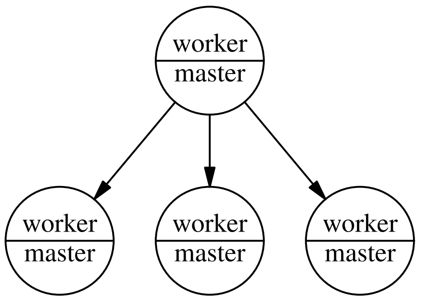

Figure 2: Our Master-Worker model. Each node can act as both a master and a worker at the same time.

2 Description of the architecture

2.1 Basic setup

Assume we need to solve an exhaustive computational problem, which can easily be broken up into multiple parts, so that each part can be computed independently of all the others. A typical problem of this kind is the enumeration of a huge number of possibilities in search of a solution.

To solve the given problem, we set up a master node named N1 and three worker nodes named N2, N3 and N4. For the reasons that will become obvious later, we assume that nodes N1, N2 and N3 are in the same LAN, whereas N4 resides in a remote network. This setup is shown in Figure 3. The arrows are pointing from the master towards the workers to indicate how the work is being assigned.

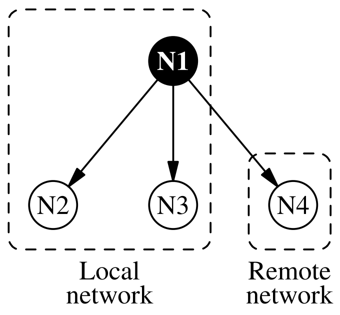

Figure 3: A basic setup with four nodes. N1 acts as a master for nodes N2, N3 and N4. N4 resides on a separate network from the other three nodes.

2.2 The process of computation

The computation begins when the user assigns master node N1 the total amount of work W, which is to be performed by the distributed system. Since N1 has three worker nodes and can also consider itself a worker, it proceeds to divide the work among all four nodes. To maximize efficiency, the work is distributed evenly, so that all nodes complete the computation within the same period of time.

To determine the amount of work, which will be assigned to each individual node, N1 measures its own performance and the performance of N2, N3 and N4. This is accomplished by running benchmarks on all four nodes and measuring the respective execution times. Instead of relying on standard benchmarks (e.g. measuring FLOPS or MIPS), the nodes are assigned a small fraction of the work they will be performing soon to solve the given computational problem. The results from these benchmarks provide a much more realistic measure of their performance in the context of solving the specific problem at hand.

The quantitative measure of the performance of the node N_i is expressed as its power P_i, which corresponds to the amount of work W_i this node can perform in a given time:

$$P_i = \frac{W_i}{t_i} \tag{1}$$

For the purpose of benchmarking, N1 assigns itself and each of its workers the same amount of work W_b. The node N_i completes the given work in time t_i, which allows N1 to calculate the power of each node using equation (1). The total power of the whole distributed system is given by the sum of the powers of its individual nodes:

$$P = \sum_{i=1}^{4} P_i \tag{2}$$

Since N1 strives to divide the total work W in such a manner that all the nodes finish their work in the same period of time, it uses the measured powers P_i , $i \in \{1, 2, 3, 4\}$ and based on equations (1) and (2) calculates the amount of work W_i as follows:

$$W_i = \frac{P_i}{P} W \tag{3}$$

2.3 Extending the system

As we have mentioned in the introduction, each node in our system is dual in nature. Therefore, each of the worker nodes, N2, N3 and N4, can also act as a master for another set of workers.

Imagine that the administrators of the remote network, which N4 is a part of, got three spare computers, which they would like to make available for our computations. These three computers reside in the private part of the remote network and do not have access to N1. The same three computers, however, can communicate with N4, so they can be attached to our tree as worker nodes of N4. The scenario is shown in Figure 4 with the new worker nodes labeled N5, N6 and N7.

Figures 4, 5 and 6 illustrate two most important properties of our distributed computing architecture:

1. Despite the fact that three new nodes have been added to the system, the original master N1 doesn't need to care about it. N1 still communicates only with its own worker nodes and perceives the subtree rooted at N4 as a single powerful computer (Figure 5).

2. When N4 receives a piece of work from N1, it acts as an independent master and distributes this work evenly among itself and its worker nodes, N5, N6 and N7 (Figure 6).

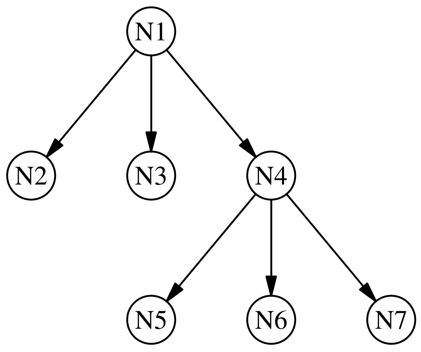

Figure 4: N4 has become a master for a set of three worker nodes.

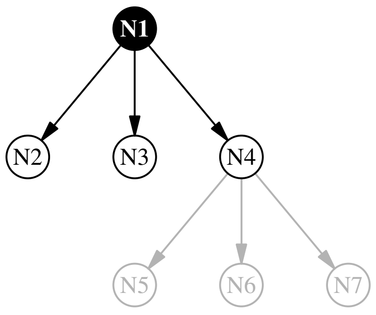

Figure 5: Node N1 still "sees" only nodes N2, N3 and N4. It is not aware of the workers of N4.

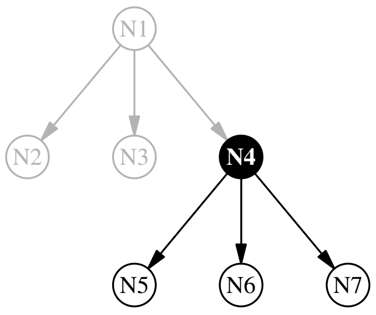

Figure 6: Once N4 gets a piece of work from N1, it acts independently distributing work among its workers.

2.4 Work distribution in the augmented tree

Let us now analyze how work is distributed in the tree shown in Figure 4.

N1 begins by running benchmarks to determine its own power and the power of its worker nodes. As we know, it does so by assigning each of the nodes N1, N2, N3 and N4 the work W_b. This time, however, N4 will share the work W_b with its workers and will therefore complete the benchmark in less time than before (i.e. when it had no workers).

Furthermore, since N4 is now a master too, it performs the same kind of benchmarks as N1 to determine the power of itself and each of its workers. This allows it to evenly distribute the work it receives from N1.

If by P_i^j we designate the power of node N_i as perceived by node N_j, then we can write:

$$P_4^1 = \sum_{i=4}^{7} P_i \tag{4}$$

In other words, N1 now perceives N4 as a single powerful computer whose power is equal to the sum of individual powers of nodes N4, N5, N6 and N7. Consequently, N1 will assign N4 a greater amount of work W_4^1, which is proportional to the perceived power P_4^1:

$$W_4^1 = \frac{P_4^1}{P} W \tag{5}$$

The total power P of the whole distributed system is again calculated by N1 as the sum of the powers of itself and its workers:

$$P = P_1 + P_2 + P_3 + P_4^1 = \sum_{i=1}^{3} P_i + \sum_{i=4}^{7} P_i = \sum_{i=1}^{7} P_i \tag{6}$$

When N4 receives the work W_4^1 from N1, it calculates the amount W_i to assign to each of the nodes N4, N5, N6 and N7, using equation (3) and the powers P_i it determined by running its benchmarks:

$$W_i = \frac{P_i}{P_4^1} W_4^1 \ , \ i \in \{4, 5, 6, 7\} \tag{7}$$

2.4.1 A simple example

Let us assume that all the nodes in our tree are computationally equally powerful. In that case the total amount of work W should be divided into seven equal parts and each part assigned to a single node. This happens in the following way:

After running the benchmarks on N1, N2, N3 and N4, N1 learns that $P_2 = P_3 = P_1$ and $P_4 = 4P_1$. Therefore, N1 will distribute the total work W as shown in Figure 7. From its own benchmarks, N4 will have learned that each of its workers has the same power as itself, so when it receives the work $\frac{4}{7}W$ from N1, it will divide it into four equal parts and assign itself and each of its workers exactly $\frac{1}{7}W$. Figure 8 illustrates this.

2.5 Dynamic work distribution

If the whole work W is divided and distributed only once – at the beginning of the computation – it becomes difficult to add new nodes to the system, remove existing ones, or restructure the tree in any other way. In exhaustive computations, which take days, weeks or even months to complete, it is

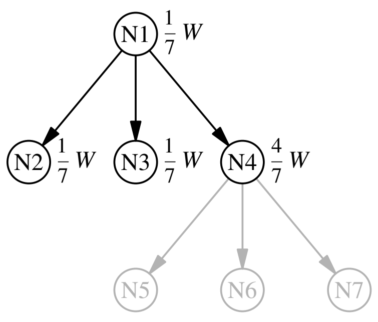

Figure 7: The total work W as distributed by N1

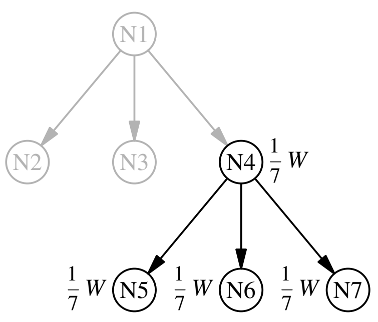

Figure 8: The $\frac{4}{7}$ of total work W after being distributed by N4

highly desirable to have the ability to add more computing power "on-the-fly" while the computation is running. Other situations may require it to take some of the nodes offline, and doing that without disturbing the running computation is a very useful feature.

In order for our distributed system to become easily adaptable to all sorts of changes in the structure of the tree, we employ the following straightforward approach:

Instead of distributing all the work at once, the master node N1 assigns each of its workers only a small fraction of W. Again, the work is distributed evenly, so that all the nodes complete their part of work within the same time. When the work is done, the workers deliver their results back to N1. N1 then assigns each of its workers a new fraction of work, and the whole process is repeated until all the work W is complete.

One important benefit of assigning the work in small pieces lies in the fact that a master node is able to notice every change in performance of its worker nodes and adjust the amounts of assigned work accordingly.

When, for instance, node N6 is detached from our tree (Figure 9), N4 has to share its work with the remaining two workers, N5 and N7, and therefore starts delivering its results to N1 later than expected. N1 perceives this as a "loss of power" and compensates for it by reducing the amount of work it sends to N4.

A similar approach is followed when new nodes are added to the system. Assume we have access to a computer lab with four nodes, L1, L2, L3 and L4, which we can use during night hours. We set up L1 as the master and L2, L3 and L4 as its workers (Figure 10). At night, when nobody else is using the lab, the four computers are attached to our tree, so that L1 becomes the worker of N2. Once this happens, N2 begins dividing its work between itself and L1, which further distributes it among its workers. N2 is now able to perform the same amount of work in less time than before. This speedup is perceived by N1 as an increase in power of N2, so N2 starts receiving more work from N1 than before.

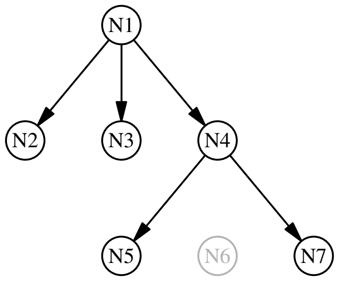

Figure 9: Our distributed system after node N6 has been removed from the tree

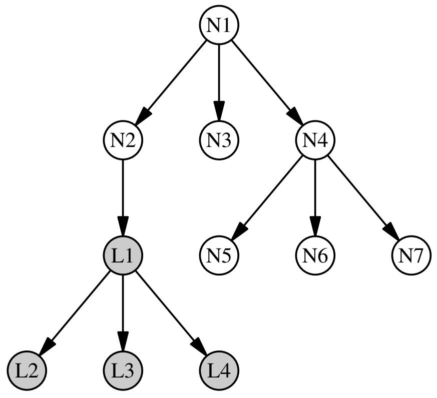

Figure 10: Our tree augmented by a computer lab with four nodes, L1, L2, L3 and L4

3 Case studies

Let us look into several scenarios which demonstrate the flexibility and scalability of the presented distributed computing architecture. We will use the sample tree from Figure 10 as a reference.

3.1 Redundancy and failover

Redundancy is particularly important for the master N1. Since every of its workers N2, N3 and N4 can act as a master too, N1 may elect one or more of these nodes as "backup" masters. Figure 11 shows a scenario, in which N2 has been chosen as the backup master for N1. Other workers, N3 and N4, are informed by N1 to "loosely" attach to N2. This means they will establish connection with N2 but won't receive any work from it. These loose connections are represented by the dashed arrows in Figure 11. If the primary master N1 fails, is taken offline by an administrator, or becomes unavailable for any other reason, N3 and N4 will activate their loose connections and become workers of N2.

Figure 12 illustrates the situation, in which the communication between N1 and N3 has failed, so N3 has become a worker of N2.

Another scenario, shown in Figure 13, happens when N1 is taken down for maintenance. In that case, both N3 and N4 become workers of N2.

3.2 Balancing the tree

Due to the nature of our architecture, the tree may become very deep with time, in which case we may experience significant delays in propagating the work from the root to the leaves. Since all the nodes in the tree are equal and dual in nature, we can reconfigure the tree in order to balance it and

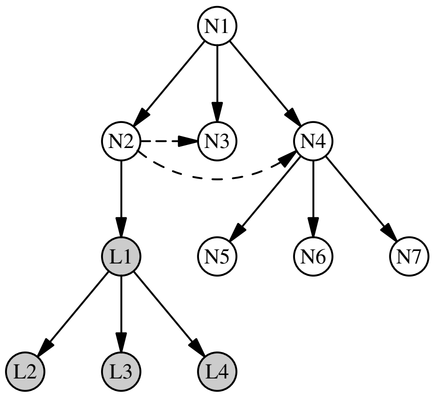

Figure 11: N2 is chosen as a backup master of N1. N3 and N4 are loosely attached to N2.

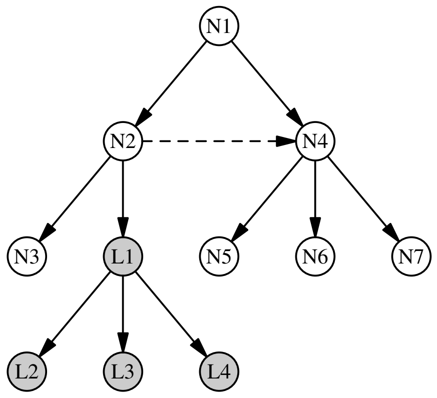

Figure 12: The communication link between N3 and N1 has failed, so N3 has become worker of the backup master N2.

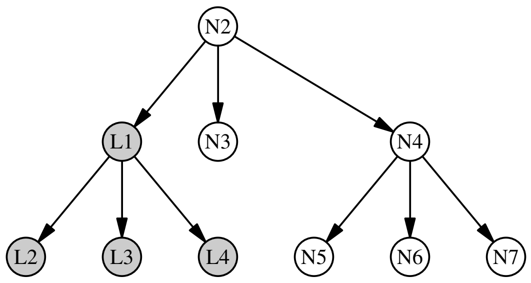

Figure 13: After N1 has been taken offline, N2 assumes its role and becomes master for the nodes N3 and N4.

decrease the latency (Figure 14). This reconfiguration could be performed manually or automatically (e.g. after measuring the depth of the tree).

3.3 Offline computation with result caching

In certain circumstances, it might be difficult or impossible to maintain a connection between a master and one or more of its workers. In such cases, the master may assign a bigger amount of work to these workers, and then let them disconnect. Each of the nodes can perform the computation by itself or as a master together with its own workers (Figure 15). The results obtained from these offline computations can be cached and delivered to the respective masters immediately or at some later time.

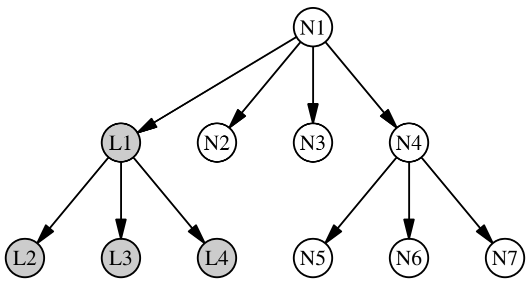

Figure 14: Balancing the tree to decrease latency. L1 is detached from its original master N2 and attached to N1.

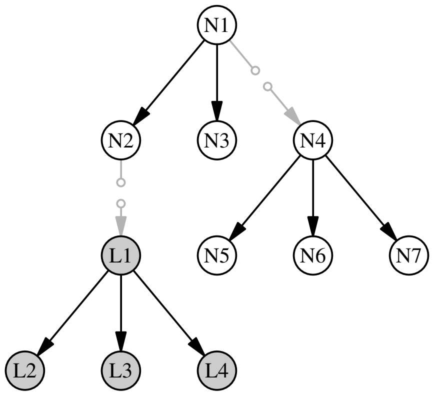

Figure 15: N4 and L1 performing work offline, detached from their masters

3.4 Attaching the whole tree to another tree

Whenever our tree is idle, we can make it available to another computational project by attaching N1 (as a worker) to a master node in the foreign tree. This feature allows for a virtually unlimited scalability.

4 Research directions

In this work we have only provided the essential functional overview of the discussed distributed architecture. There are, however, important issues which we haven't tackled and which should be subject to further research.

Among the most interesting topics for research are security (authentication and authorization), recovery from failures, scheduling of work on multiprocessor nodes, handling communication delays in large distributed systems, and methods for work distribution in environments with limited connectivity (e.g. delivering work by e-mail).