1986

On Distributions of Run-Times in Distributed Systems

Vernon J. Rego
Purdue University, rego@cs.purdue.edu

Report Number:
86-607

ON DISTRIBUTIONS OF RUN-TIMES IN DISTRIBUTED SYSTEMS

Vernon Rego

CSD-TR-607
June 1986

ON DISTRIBUTIONS OF RUN-TIMES IN DISTRIBUTED SYSTEMS

Vernon Rego
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

In distributed systems, the inherently variable processing times cannot, in general, be satisfactorily characterized purely by average or worst-case times. These measures are difficult to interpret when identical inputs show very different processing times, a situation typical of distributed algorithms executing on distributed systems. This is due to the nondeterministic nature of communication, and factors such as network load, reliability, interference, etc. Thus it becomes important to study the variance of running-times, as a function of various communication parameters. We propose a simple, yet computationally effective technique, to obtain the average running-time distribution of a distributed algorithm, involving a set of n processors on a local network.

1. Introduction

It is well-known that the *running-time* of an algorithm (generally) depends on the *size* and *nature* of its input. An algorithm's behaviour is usually characterized by its average and worst case running times, for a given input of size n. Such a course is adequate for algorithms and systems exhibiting low variability in processing times, and it can be argued that many known algorithms fall in this class. Besides this argument, average (and worst case) times are difficult enough to obtain and usually rely on such assumptions as equally likely inputs, perfectly random behaviour, etc. However, an approach that attempts to characterize a distributed algorithm's behaviour (i.e., its running-time on a distributed system) by average and worst-case times fails to capture the very essence of distributed system behaviour, namely, *high variability.*

Consider a situation in which M general purpose processors (units) communicate with one another over a local network. If n, $n \leq M$, of these units are simultaneously given a problem at time t_i, and the solution to the problem is obtainable only via mutual cooperation, then the units must exchange information over the network (using a mutually-agreed upon distributed algorithm), in order that the problem is finally solved at some time t_f. The random time $T = t_f - t_i$ will depend on various things, such as the speed and type of local network, speed of the processors, reliability, choice of distributed algorithm, nature of the problem, size of n, load on the network, etc. In such a situation, it is clear that running-time is highly variable, since variability is an inherent property of distributed system behaviour, and thus running-time is very poorly characterized by average or worst-case measures. For example, running-times can be very different *for the same input*, simply because of network traffic or related phenomena. A mean value for running-time cannot describe this aspect of the algorithm's behaviour.

Applications involving such problems arise in distributed databases, distributed computing, network protocols, etc. The exchange of data between computing units chiefly depends on

(1) the behaviour of the local network (or equivalent) allowing communication between the computing units, and

(2) the type of algorithms used by the different computing units, both within units and between units.

In this note, we are interested in the distribution of time required to solve a given problem by cooperating, but general purpose processors. As a specific example, we choose to work with a sorting algorithm on an abstract network. Such an algorithm has the nice property of being simple, yet sufficient to demonstrate the effects of high variability of processing times on distributed systems.

2. The problem definition

Consider a system of n distributed processors that communicate with one another over a local network, with the intention of solving a given problem. Suppose that each processor j in the set of processors $\mathbf{S} = \{1,2,...,n\}$ is simultaneously loaded with an integer x_j, such that $x_j \in \mathbf{S}$, but $j \neq x_j$. We assume that the assignment of integers to processors is random, and each configuration (of assignments) is as likely to occur as every other configuration. As is usually the case with local networks, we assume that processors are connected over a channel, such as in a ring or bus network. At any given time, only one processor may successfully utilize the channel.

Given a configuration of integer assignments to the various processors in S, we say that processors i and j are *unordered* if $i < j$ and $x_i > x_j$. Let W represent the set of initially unordered processors, W a subset of S. A distributed algorithm to *order* the processors is now executed, so that the set W can be reduced to an empty set by step-wise reduction. When W is empty, we say that the system is *ordered.* If at any time $|\mathbf{W}| = k$, then the ordering problem is of size k, and denoted by $P(k)$.

Let $g(n)$ denote the average running time of the sorting algorithm for $P(n)$ given that it is

executed on a single processor. This is equivalent to saying that $g(n)$ is the time taken by the distributed algorithm given that all communication on the network is perfectly reliable, overheads of communication delay are nonexistent, and only one processor may use the communication channel at any time. For example, if the distributed sorting algorithm is essentially a *bubblesort*, then the algorithm's contribution to $g(n)$ is $O(n^2)$.

In the context of distributed control, observe that in using a bubble-sort like algorithm, each *swap* no longer takes a constant amount of time (as is the case with sorting in an array). The time taken by two processors to swap information is highly dependent on the type of local network, medium access protocol, network loads, reliability, mean response times, etc. As an example, if the network is a ring network then the mean swap time between two processors is roughly given by the sum of the expected values of the response times of the two processors involved in the swap. In order to incorporate reliability, we can say that the swap is successful with a specified probability r, and unsuccessful with a probability $(1-r)$. In the latter case, the two processors must try to swap again (possibly via an alternate route), thus effectively increasing the swap time.

The highly nondeterministic nature of processing-times in distributed communication allows us to make the simplifying assumption that the entire system (which is simultaneously doing many things besides solving this sorting problem) spends a geometrically distributed random time in reducing $P(k)$ to $P(k-1)$, $1 \leq k \leq n$. The random time T_k spent by the system in reducing $P(k)$ to $P(k-1)$ is assumed to have parameter $p_k = r_k/g(k)$, for $k = N,...,1$. Here r_k is the probability that each of the processors involved in solving $P(k)$ will use its network "slot" (i.e., chance to successfully capture and use the channel) towards solving $P(k)$. Thus $(1-r_k)$ is the probability that a processor involved in solving $P(k)$ will use its network slot towards something else, and not involving $P(k)$. When this happens, it naturally takes longer to solve $P(k)$. If the sorting problem is the only load on the system (i.e., an unlikely situation in a local network, for this means the system is doing nothing else), then $r_k = 1$ for each $k \in \mathbf{S}$.

Observe that for larger values of k the parameter r_k will tend to be smaller, meaning that it will generally take a longer time to reduce $P(k)$ to $P(k-1)$ when k is larger. Correspondingly, r_k larger when k is smaller, so that it generally will take a smaller amount of time to solve $P(k)$ when k is smaller. This can be attributed to the amount of communication involved. When k is large, more processors are involved in communicating distributed information, and thus the time taken for them to reach some form of mutual agreement is larger. Not only is network load higher now, but so is the variability of configurations of relevant (for our algorithm) events. When k is smaller, this time naturally becomes smaller.

3. Run-time Distribution

The distributed algorithm to solve $P(n)$ can be viewed as a system working in phases. In the initial phase, possibly n processors will have their integers out of order. The processors decide on the kind of algorithm they will use and then begin to execute it by exchanging information. After a certain period of time (depending on the choice of their algorithm) possibly $(n-1)$ processors will have their integers out of order. This begins the second phase. Recall that each phase takes a geometrically distributed amount of time. Even if two or more processors manage to arrange themselves (i.e., sort themselves based on their respective integers) during the same phase, we can still assume that this takes place in different phases via a steady-state "averaging" argument. This yields a run-time distribution based on the uni-processor average running-time $g(n)$. Thus, exactly k phases will be required to solve the problem $P(k), k \leq n$.

The probability that there are $(n-k)$ processors with integers initially out of order is given by

$$Pr[|\mathbf{W}| = n-k] = Pr[|\mathbf{S}-\mathbf{W}| = k] = \frac{1}{k!}[1 - \frac{1}{1!} + \frac{1}{2!} + \cdots + (-1)^{n-k}\frac{1}{(n-k)!} \quad (1)$$

for $k = 0, 1, 2, \cdots, n$. Observe that $Pr[|W| = 1] = 0$. This is known as a matching distribution, obtained via an application of Boole's formula.

Given that there are $(n-k)$ processors with integers initially out of order, it takes $(n-k)$ phases of geometrically distributed times for them to sort themselves in order. Due to the goemetrically distributed amount of time that the system spends in each phase, the cooperating set of processors behave in Markov fashion. The system remains in phase 1 with probability $(1-\hat{p}_{n-k}) = 1-\frac{r_{n-k}}{g(n-k)}$ and moves to phase 2 with probability $\hat{p}_{n-k}$. In phase 2 there are $[n-(k+1)]$, processors out of order. The system remains in phase 2 with probability $(1-\hat{p}_{n-(k+1)})$ and moves to phase 3 with probability $\hat{p}_{n-(k+1)}$. When the system of processors finally leaves phase $(n-k)$, we say that the distributed algorithm has run to completion and $P(n-k)$ is solved.

Given $(n-k)$ processors with integers out of order, we can define an $(n-k+1)$ state Markov chain of the form

$$\mathbf{P}(n-k+1) = \begin{vmatrix} \mathbf{T}_{n-k} & \mathbf{T}^{0}_{n-k} \\ 0 & 1 \end{vmatrix} \tag{2}$$

where

$$\mathbf{T}_{n-k} = \begin{array}{c} \\ 1 \\ 2 \\ \\ \\ n-k \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & \cdot & \cdot & n-k \end{array} \\ \begin{bmatrix} (1-\hat{p}_{n-k}) & \hat{p}_{n-k} & & & & \\ & (1-\hat{p}_{n-(k+1)}) & \hat{p}_{n-(k+1)} & & & \\ & & & & & \\ & & & & & \\ & & & & & (1-\hat{p}_1) \end{bmatrix} \end{array}$$

is a substochastic matrix, with $\mathbf{I}-\mathbf{T}_{n-k}$ nonsingular, and $\mathbf{T}_{n-k}\mathbf{e} + \mathbf{T}^{0}_{n-k} = 1$. The vector e is the unit vector with all entries set to unity. The matrix **P** in (2) is used to define the well-known phase-type distribution [1]. Given that the system begins communicating to solve $P(n-k)$ (i.e., in phase 1), we use $\alpha = (1, 0, 0, \cdots, 0)$, an $(n-k)$ vector, as the *initial probability vector*, to obtain the probability density $\{p_m\}$ of the number of steps taken by the distributed algorithm to complete its execution. The phase-density [1] is given by

$$p_0 = 0, \quad \text{and}$$
$$p_m = \alpha \mathrm{T}_{n-k}^{m-1} \mathrm{T}_{n-k}^{o}, \text{ for } m \geq 1. \tag{3}$$

Using (1) and the fact that finite mixtures of phase-type distributions are also phase-type distributions, the running-time distribution of the distributed algorithm is also of phase-type. If the number of steps required by the distributed algorithm to run to completion is d, then the probability that the algorithm runs to completion in m steps is given by

$$Pr[d = m] = \sum_{k=0}^{n} Pr[|W| = n - k]\, p_k \tag{4}$$

where for each k, $\{p_k\}$ is appropriately defined via $\mathbf{T}_k$.

4. Summary

A simple, yet computationally effective technique for computing the running-time distribution of a distributed algorithm was presented. We chose to work with a sorting algorithm on an abstract network to demonstrate the general idea. Since the problem in all its generality is basically intractable, we make an assumption that the system spends a geometrically distributed amount of time in solving each phase of the entire problem. That is assumption is reasonable follows from the fact that each time a processor has a chance to use the channel and do its bit towards solving the current subproblem, a variety of things might happen. The processor might crash temporarily, or it might use the network to perform another task (solving some other problem that has a higher priority), or it may communicate with another processor in a manner that prolongs the current solution phase. The idea presented might be extended to a number of situations. It would be of some interest to examine the performance of a given algorithm on different kinds of networks, since the type of network (and loads) is bound to affect the algorithm's behaviour.

References

[1] Marcel F. Neuts, *Matrix-Geometric Solutions in Stochastic Models*, The Johns Hopkins University Press, 1981.