PROBABILISTIC BIDDING GIVES OPTIMAL DISTRIBUTED RESOURCE ALLOCATION

John H. Reif

Paul Spirakis

TR-23-83

***Preprint of paper appearing in Lecture Notes in Computer Science*, Vol. 172, pp. 391-402.**

PROBABILISTIC BIDDING GIVES OPTIMAL DISTRIBUTED RESOURCE ALLOCATION†

John Reif

Aiken Computation Laboratory,
Harvard University
Cambridge, MA 02138

Paul Spirakis

Courant Institute of Mathematical Sciences,
New York University

and

Computer Technology Institute
Greece

ABSTRACT

In this paper we consider a fundamental problem of *resource allocation* in a distributed network. Informally, the problem is to concurrently satisfy many processors requests for resources, each of which can be used by only one process at a time. In paricular, a processor may simultaneously issue requests for various resources. It then awaits a grant for all its requested resources. Upon receiving the resources, the processors may use them for only a limited period of time. Finally, the process must reurn these resources. This must proceed concurrently for all requesting processors. We seek distributed algorithms, which satisfy dynamically changing user requests for resource allocation, in a distributed way, by only a local communication between granting and requesting processes.

Each user's demands for resources may change dynamically and the processors speeds can vary dynamically. Let v be the maximum number of users competing for a particular resource at any time instant. Let k be the maximum number of resources that a user is wil-

† This work was supported in part by the National Science Foundation Grants NSF-MCS79-21024, NSF-MCS-8300630, DCR-85503497, and the Office of Naval Research Contract N00014-80-C-0647, and the Greek Ministry of Science and Technology. This paper appeared, in a preliminary form, in the *11th International Colloquium on Automata, Languages and Programming*, July 1984, Antwerp, Belgium.

ling to get, at any time instant. This problem was previously formulated in [Lynch, 1980]. It has application (1) to two-phase locking in databases (2) to generalized dining philosophers, and (3) to the implementaion of a novel extension of the CSP language, called Social-CSP and to many other applications to concurrent programming.

Informally we say that an algorithm for this problem is *real time* if its response time is upper bounded by a function which does not depend on any global measure of the system of processes and resources, except k and v. [Reif, Spirakis, 1982b] gave the first known real time algorithms to the problem with (mean) response time of $O(v^k)$. This response time may be too long, however, in applications where k has a large value.

In this paper we provide new algorithms whose response time is *polynomial* in v and k. Our most efficient new algorithm has expected response time $O(vk)$. Moreover, our constant factors appear to be small enough for practical applications.

Unlike our previous probabilistic algorithms of [Reif, Spirakis, 1982b], we do not use random delays as means of avoiding process starvaion and of achieving probabilistic fairness. Instead, our new algorithm utilizes a method of *probabilistic bidding* to resolve contention of users for resources. This technique is essential in our achievement of polynomial response time.

We furthermore prove that our solution is *optimal* with respect to the average response to a user's request. In particular we provide matching *lower bounds* for any distributed algorithm for resource allocation, and these bounds are within a constant factor of the response time of our own algorithm.

We also provide a suboptimal algorithm, which allows user processes which demand less than k resources to have higher probability of being assigned. This algorithm has the property that if a user i has a request of at most $k_i \leq k$ resources, then it has mean response $O(vk_i \log k \log(k_i v))$. This suboptimal algorithm does use random waits combined with probabilistic bidding. These techniques employ limied parallelism within each process, together with the probabilistic bidding. (This limited parallelism is useful in achieving optimal response time, though we would still get polynomial response time without limited parallelism.)

1. INTRODUCTION

1.1. Resource Granting Systems.

In this paper, we consider a resource allocation problem previously described in [Lynch, 1980] and generalized in [Reif, Spirakis, 1982b]. The system has potentially an infinite set of processes π and each process has an integer name. There is a potentially infinite set of resources ρ in the system. Let $R \subseteq \pi$ be the set of processes which control resources. The set of user's processes is $U = \pi - R$. Each resource $\rho(j) \in \rho$ is controlled by a distinct granting process j. This process has the responsibility of granting each resource to a distinct user process in U, so no two different processes in U may use the same resource at the same time. We assume that there is a global time which totally orders events, but processes may not have access to the global time. Users' processes communicate only with those granting processes for which they request resources.

A system as above is called a *Resource Granting System* (RGS); see [Reif, Spirakis, 1982b]. The goal of an RGS is to satisfy dynamically changing user requests for resource allocation, in a distributed way, by only a local communication between granting and requesting processes.

The set of possible schedules of processes actions is determined by an "adversary" oracle **A** which has the power to set actions in the worst possible way to increase the response time. **A** has also the capability to select at time $t = 0$ the schedule of speeds of all processes at all times $t \geq 0$. In addition, at each time $t \geq 0$, the requests by user processes U are specified by **A**. The adverse oracle **A** is restricted, to allow users to keep asking for their resources until they are granted. In practice, no such **A** may exist but coincidence of worst case situations may replace is action.

Let a process be *tame* during a time interval Δ, if for any interval Δ' which intersects Δ and is a single step of the process, then $|\Delta'| \in [r_{min}, r_{max}]$ where r_{min}, r_{max} are fixed real constants with $0 < r_{min} \leq r_{max}$.

We do not require processes to be tame at all times. More specifically, the basic correctness properties of our algorithm (e.g. mutual exclusion in allocation of resources) are *not compromised* in the event that the processes execute at arbitrary speeds. However, our proof that our techniques are real time makes use of the assumption that processes are tame. I.e. the nice performance and fairness properties of our algorithms depend on the tameness assumption. Moreover, even if the processes behave "untamely" for a long time, as soon as they start to behave tamely, the nice fairness and performance properies begin to hold. This is a very desirable *resilience* property for a distributed algorithm.

Finally, we assume that processes are *reliable* in the sense that they perfectly execute their programs. See Sections 1.2 and 1.3 for further details of this model and for precise definition of response time. (Sections 1.2 and 1.3 may be skipped in a first reading of the paper.)

1.2. The RGS Model

A process *step* consists of either an assignment of a variable, a test, a logical or arihmetic operation or a no-op. A step is considered to be a finite time interval Δ in which a single primitive instruction is executed just at the last moment of Δ, and no other instructions are executed within Δ. Let a process be *tame* during a time interval Δ, if for any interval Δ' which intersects Δ and is a single step of the process, then $|\Delta'| \in [r_{min}, r_{max}]$ where r_{min}, r_{max} are fixed real constants with $0 < r_{min} \leq r_{max}$. We will not require processes to be tame at all times. However our proof that our algorithms are real time makes an assumption that processes are tame. We assume that processes are *reliable* in the sense that they perfectly execute their programs. The rate of execution varies dynamically. We require that, at no time, any granting process $i \in R$ simultaneously grant the resource $\rho(i)$ to more than one requesting process. We also require that, as soon as a process $j \in U$ has got all its required resources, then it can keep them only for a time interval whose length is upper bounded by a fixed parameter δ (containing at most $\mu = \delta / r_{min}$ steps, if the process is tame). Let

$\text{resources}_t(i)$ be the set of resources that process i is requesting at time instant t. Let $k_{i,t}$ be $|\text{resources}_t(i)|$. Let $\text{askers}_t(j)$ be the set of user processes requesting $\rho(j)$ for $j \in R$ at time t. Let $v_{j,t}$ be $|\text{askers}_t(j)|$. We assume that at all times $t \geq 0$, $v_{j,t}$ and $k_{i,t}$ are upper bounded by constants v,k respectively. (This does not necessarily imply any bounds on $|\cup_{t=0}^{\infty} \text{resources}_t(i)|$ or $|\cup_{t=0}^{\infty} \text{askers}_t(j)|$ for any i in U or j in R).

With respect to interprocess communication we assume (1) that each resource allocator $j \in R$ has a set S_j available to it of size at most v, containing the names of those processes willing to get the resource. As in [Lynch, 1980] we assume this to be a primitive of our system (it could be implemented by a queued message system, for example), (2) that synchronization must be done by special variables, called *flags,* each of which is written only by one process and read by at most one other process. Read-write conflicts on flags are excluded by our process step semantics and by our notion of global time. Flags seem to be the simplest primitive for synchronization and lead to an easy implementation (in contrast to distributed shared variables of multiple readers and writers).

1.3. Implementations and Complexity of an RGS

An *implementation* of an RGS determines the synchronization algorithms that the processes run. As stated above, the synchronization algorithms use only flags to implement the synchronization between processes. We consider a time-varying hypergraph H_t with node set π and time-varying hyperedge set $\{\{i\} \cup \text{resources}_t(i) | i \in U\}$, i.e., where a hyperedge at time instant t is the set of nodes of π consisting of a single process i in U and the granting processes of the resources i is willing to get at t. An RGS implementation dynamically achieves distributed matchings in the hypergraph H_t.

For each adverse oracle **A**, let the *response time* of the RGS implementation be the random variable $\gamma_{A,k}$ which is the length of the smallest interval Δ required for any process $i \in U$ to have k resource requests simultaneously granted, given that i requested these resources during the entire interval Δ and assuming that, i and all allocators of the

resources requested by i within Δ, are tame within Δ.

Let the *mean response* $\bar{\gamma}_k$ be the $\max\{\text{mean}\{\gamma_{A,k}\}$ over all oracles $A\}$. Let the ϵ-*response* be the minimum $\gamma_k(\epsilon)$ such that for every oracle A $\text{prob}\{\gamma_{A,k} \leq \gamma_k(\epsilon)\} \geq 1-\epsilon$. The RGS implementation is *real time* if for all $\epsilon \in (0,1), \gamma_k(\epsilon) > 0$ and upper bounded by a function independent of any *global measure* of the network. (Note: A *global measure* of the network is any positive function g of $h = |\pi|$ such that $\lim_{h\to\infty} v/g(h) = 0$.) Hence if an RGS implementation is real time, then the mean response $\bar{\gamma}_k$ is also upper bounded by a function independent of any global measure of the network.

1.4. Previous Work

[Rabin, 1980a] first applied probabilistic choice to synchronization problems in distributed systems and provided a solution to the dining philosophers problem which, with probability 1, is deadlock free and starvation free. [Rabin, 1980b] applied probabilistic coordination methods to synchronize access to processes to a critical resource in a space-efficient manner. [Frances and Rodeh, 1980] and [Itai and Rodeh, 1981] also proposed probabilistic techniques for synchronization and leader election problems, respectively.

[Lynch, 1980] first posed the localized resource allocation problem as a formal synchronization problem. Let the resource graph G be the graph whose nodes are the resources and two resources are connected by an edge if there is ever a user process requesting both of them, maybe at different times. Let $\chi(G)$ be the chromatic number of G. The implementation proposed by Lynch was a deterministic one in which processes should know the color of each resource in a coloration of G. The worst case response time achieved in [Lynch, 1980] is of the order of $\chi(G)v^{\chi(G)}\cdot\tau$ where τ is the time necessary for interprocess communication. This was not a real time implementation since $\chi(G)$ is $\Omega(|\pi|)$ in general. (Let us note here that the Lynch algorithm behaves better than its general worst case, in special cases of systems with low congestion. Its response time is then polynomially related to $\chi(G)$, however it still is not a real time implementation.)

[Reif, Spirakis, 1982b] provided the first real time RGS implementation, with mean response time $O(kv^{k+2}\log v)$. In that previous work, we used the techniques of probabilistic selection of processes by resource allocators and random waits to avoid adverse schedules of speeds which might be set up by the oracle. Although this was a real time implementation, it was still exponential in k.

1.5. The New Results of This Paper

We shall present (in Section 3) a probabilistic implementation of an RGS, with mean response $O(kv)$ and ϵ-response $O(kv \log(1/\epsilon))$. Not only is the expected response of our implementation is upper bounded by a constant, but also the *actual time* needed by any user to get its requested resources, is upper bounded by a constant with overwhelming probability. Violations of this property occur with vanishingly low likelihood. Furthermore, we make *no* probability assumptions about system behavior. Because of the above facts, we feel that our algorithm compares favorably to the worst case performance of previously developed deterministic algorithms for the same problem.

To achieve the stated response time, we make essential use of the probabilistic bidding technique, together with use of limited parallelism within each user process and each resource allocator. In our *uniform bidding* algorithm, we do not use random waits to achieve probabilistic fairness. Instead we use only the probabilistic bidding technique. In particular, we slice the time of each process into rounds. In each round each user process tries to get all the wanted resources. It has to get all of them in the same round. The users do not accept partial allocation of resources to them, unless *all* the required resources are offered to be allocated in a small number of steps. At the end of the round, users release their allocated resources (if any) and make a fresh start. User rounds have the same length in steps for all users and this length is a parameter of the algorithm. The time of the granting processes is also sliced into rounds (resource rounds). In contrast to user rounds, resource rounds are not of the same length and their length in steps is not fixed in advance, but adjusts to the conditions of the

algorithms. We conjecture that this is essential in avoiding exponential growth of the response with k.

We also prove lower bounds of $\Omega(kv)$ for the worst case and average response time of any algorithm for the local resource allocation problem. Thus our proposed technique is of optimal performance within a constant factor. We also provide a *priority bidding* algorithm which has mean response time polynomial in k (however not optimal) and is useful for improving the throughput rate of resource allocations in the network. In particular, it allows user processes which demand less than k resources to have higher probability of being assigned. This algorithm has the property that if a user $i \in U$ has a request of at most $k_i \leq k$ resources, then it has mean response $O(vk_i \log k \log(k_i v))$.

1.6. Applications

Example 1: Lefty k Philosophers. As a simple example of the usefulness of RGS, consider a generalization of the dining philosophers problem to the case where each philosopher requires k-forks to eat. (This problem was first considered in [Lynch, 1980].) We extend it to the case where the identities of the forks required by each philosopher change dynamically. Let the set of "forks" be $R = \{r_0,...,r_{n-1}\}$ and the set of "philosophers" be $U = \{u_0,...,u_{n-1}\}$ and let $\text{resources}_t(u_i) = \{r_i, r_{(i+1) \bmod n},...,r_{(i+k-1) \bmod n}\}$ and $\text{askers}_t(r_i) = \{u_{(i-k+1) \bmod n},...,u_{(i-1) \bmod n}, u_i\}$ for all t. Our new resource allocation algorithm achieves mean response time $O(k^2)$. In contrast, our previous results achieved mean response time $O(k^{k+3})$, (see [Reif, Spirakis, 1982b]).

Example 2: Social CSP. An extension of CSP, defined and discussed in [Francez, Reif, 1985], has an efficient implementation by our real time RGS. Social-CSP has the following new commands:

(1) *Extended Output Command:* $(p_{j_1},...,p_{j_k})!(u_1,...,u_k)$ in which the sender process simultaneously sends the value u_m to process p_{j_m}, $m = 1,...,k$. Here, "simultaneously" means that the receipt of a value by a process named in the output command does not affect in any way the receipt of the values by other processes named in the output command. Note that (1) can

be considered as the generalization of a broadcast command.

(2) *Extended Input Command:* $(p_{i_1},...,p_{i_k})?(x_1,...,x_k)$ where the receiver process simultaneously gets a value for its variable x_m from process p_{i_m}, $m = 1,...,k$.

Although these extended input and output commands can, in theory, be simulated in Hoare's CSP, it is not clear how to provide an efficient simulation. The power of the new constructs of Social-CSP can be demonstrated by the simplicity they give to a program solving the k-fork philosophers problem. In contrast, it is not known how to solve the k-fork philosophers problem by the conventional CSP constructs.

Social-CSP commands can be directly implemented by our RGS real time implementation, by considering the sender in the output command (respectively the receiver in the input command) as a user process and the processes $p_{j_1},...,p_{j_k}$ (respectively $p_{i_1},...,p_{i_k}$) as resource granting processes. Note that our implementation of Social-CSP allows for unspecified or computed targets of communication, since the identities of the resources a user wants may change dynamically. (This is useful in case of routing protocols and was first considered in [Francez, 1982].)

Example 8: Two-Phase Locking in Databases. Two-phase locking is a concurrency control method in databases; for a survey see [Bernstein, Goodman, 1980]. It has the feature that as soon as a transaction releases a lock, it never obtains additional locks. A very efficent static implementation of two-phase locking can be achieved by our methods. Our assumption is that transactions are allowed to act on the data only if they have already obtained all the locks requested. In the context of such a database system, let the users in U be called *transaction modules* and the processes of R be called *data modules*. If the readsets of the transactions are of cardinality at most k at each time instant and if at most v transactions can compete for a lock at a time instant t, then our optimal RGS will result in an $0(vk)$ mean response time per transaction. Our suboptimal RGS achieves an even smaller mean response time when $|\text{readset}_i(t)| = o(k)$. In this case, if a transaction wants to lock k_i data items at a time, it has a mean response $0(vk_i \log(vk_i)\log k)$. (However, this becomes

$0(vk \log(vk) \log k)$ when $|\text{readset}_i(t)| = k$.) Our implementations of two phase locking proposed in this paper are asymptotically more efficient than the static locking method proposed in [Reif, Spirakis, 1982b], which had a mean response $0(kv^{k+2})$. Thus our new algorithm becomes advantageous in cases of database systems with small granularity of locking and hence very large cardinality of transactions readsets. In those cases other known algorithms are impractical since they have response times exponential in k.

2. AN $\Omega(kv)$ LOWER BOUND FOR THE LOCAL RESOURCE ALLOCATION PROBLEM

THEOREM 1. *For $k > 0$ and $v = k$ there is a network in which at least one user process has two have a response time of at least $((kv/4) - 1)\mu$ steps.*

Proof. Consider a network with a set of users U such that $|U| = a \cdot b$, where $a = b = \lceil k/2 \rceil = \lceil v/2 \rceil$ (see Figure 1). The set of resources R has $|R| = ab$ also, and let us represent it as a matrix of elements $\rho_{11}, \ldots, \rho_{1a}, \ldots, \rho_{ba}$. Let $\mathbf{A}$ be an oracle such that all processes are equi-speed, synchronous and such that for all $t \geq 0$ and for all $j \in U$, if $j = am + n$ (where $0 \leq m \leq b-1$ and $1 \leq n \leq a$) then

$$\text{resources}_t(t) = \{\rho_{m+1,1}, \rho_{m+1,2}, \ldots, \rho_{m+1,a}\} \cup \{\rho_{1n}, \rho_{2n}, \ldots, \rho_{bn}\}$$

i.e. the $m+1$-th row and the n-th column of the matrix R. Then, only one user process can be granted all its resources at each time instant t, simply because any two pairs of a row and a column of R each, intersect in at least 2 resources (and each resource has to be granted to only one user at a time, thus forming a bipartite matching of the hypergraph defined in Section 1.3). This implies that resources will be allocated to users *serially,* hence the last process in the serial order will have a response time of at least $(ab - 1)\mu$ steps, i.e. of at least $((kv/4) - 1)\mu$ steps.

Note also that the maximum demand per user is $a + b - 1 \leq k$ and that the maximum number of users competing for a particular resource is also $a + b - 1 \leq v$.

Finally, let us note that the above proof *does not rely on* any complicated adverse timing of requests, and holds independently of the synchronization technique. □

THEOREM 2. *For $k > 0$ and $v > 0$ there is a network in which at least one user process has to have a response time of at least $((kv/4) - 1)\mu$ steps.*

Proof. Consider a network with a set of users U, such that $|U| = ab$ with $a = \lceil k/2 \rceil$, $b = \lfloor v/2 \rfloor$. Let us partition the users into a groups, such that group i consists of the users $u_{i-1+1}, u_{i-1+2}, \ldots, u_{i-1+b}$, for $1 \leq i \leq a$. (We consider here an enumeration $u_1, u_2, \ldots, u_{ab}$ of the user set U.)

The network has also a resource set R of $|R| = a^2$. We consider R to be an $a \times a$ matrix of resources $[\rho_{i,j}]$ where $1 \leq i \leq a$ and $1 \leq j \leq a$ (see Figure 2).

To complete the network description, let $\mathbf{A}$ be an oracle such that all processes are equi-speed, synchronous and such that for every $t \geq 0$ and every $u_j \in U$: if u_j belongs to group g, then he set $\text{resources}_t(j)$ is the union of row g and column g of the matrix R. Then, only one user process can be granted all its resources at each time instant t, because any two pairs, of a row and a column of R each, intersect on at least 2 resources, and each resource has to be granted to only one user at a time. This implies that resources will be allocated to users *serially,* hence the last process of the serial order will have a response time of at least $(ab - 1)\mu$ steps i.e. of at least $((kv/4) - 1)\mu$ steps.

Note that the maximum demand of resources per user is $2a - 1 \leq k$ and that the maximum number of users competing for resource $\rho_{i,j}$ is b (because of group i) plus b (because of group j) i.e. $2b \leq v$. This holds for any resource $\rho_{i,j}$.

Finally, let us note that the above proof *does not rely on* any complicated adverse timing of requests and holds independently of the synchronization technique that a possible implementation could use. □

COROLLARY. *Our probabilistic bidding algorithm of Section 3 has optimal mean response within a constant factor.*

Proof. By Theorem 2 and by the fact that given any multiset of serial orders of $y = kv/4$ elements, there is at least one element whose *average* position (over the multiset of orders) is at least $\lfloor y/2 \rfloor = \lfloor kv/8 \rfloor$. □

3. OUR DISTRIBUTED UNIFORM PROBABILISTIC BIDDING ALGORITHM

We assume that the requesting processes communicate only to the resource allocators whose resources they want (or have been allocated), and that each granting process j is willing to communicate only to the requesting processes in the set S_j (as defined in Section 1.2). The actions of the requesting and granting processes are time-sliced in *rounds,* each round being a repetition of a basic set of actions. Processes use independent sequences of probabilistic choices as the basic construct to counteract adverse speed schedules and adverse resource demands set up by the oracle **A**. We assume that **A** cannot affect or foresee the results of these probabilistic choices. We allow each user in U and each resource allocator in R to have a set of synchronous parallel subprocesses, which aid in our algorithms. The use of local parallelism here is not actually essential in achievement of polynomial response time.

3.1. An Informal Description of the Rounds

a. *The User's Round.* A user's round starts with the user drawing (with equal probability) a random number in the set $\{1,2,...,\beta kv\}$ where $\beta \geq 1$ is an integer. If the number drawn was less than βkv, the use remains nonactive, until the end of the round.† (All users' rounds take a predetermined number of steps.) Else, the user immediately notifies (by the use of at most k parallel synchronous subprocesses) all the resource allocators of the resources he wants, that he is a winner. Then, the user's parallel subprocesses collect answers from the resources for a period which is bounded by a constant number of steps. During this period some of the resources may declare that they agree to be allocated to the particular user. However, if at that time, any other resource requested by that user is denied, then that user does not utilize the resources which agreed to be allocated to him, but he continues to report

---

An alternative way to implement the random choice is to use an unfair two-faced coin of Prob{success} = $1/\beta kv$. In the case of failure, the user stays nonactive, until the end of the round. The two alernatives are comparable in terms of implementation difficulty.

that he is a winner to all of his requested resources and repeats the algorithm (without drawing again), until the user's round ends. If all of the wanted resources agree to be allocated at the same period (in which the user collects answers), then the user utilizes them for μ steps (μ is a small integer constant, as in Section 2.1) and then he releases these resources, using his k subprocesses. Note that a communication with all the k resource allocators takes only r_{max} time due to the limited parallelism and tameness of processes.

b. *The Resource Allocator's Round.* The round of resource allocator j starts with a *monitoring period* of a constant number of steps during which at most v parallel synchronous subprocesses continuously monitor the users of the set S_j, looking for winners. Let M_j be the se of winners detected during the monitoring period. If M_j contains more than one winner, then all the elements of M_j are notified in parallel that they have been denied, and the round ends. However, if M_j has a unique winner, then the granting process notifies the winner that it agrees to be allocated. If the winner does not accept the agreement then the rounds ends. If the winner accepts, then the round enters an *allocation period.* During this period, the parallel subprocesses of the resource allocator deny all appearing winners. The round now ends by receipt of the notification by the user that the resource has been released.

c. *Additional Remarks.* Note that communication with all v of the user processes and all set operations in a resource allocator's round take only a constant (independent of v and k) number of steps due to the parallelism employed. Note also that the following holds with certainty:

A resource decides to be allocated to a *unique* winner, only after the resource allocator agrees to allocate the resource and the winner accepts the agreement. Thus, no resource can be allocated to more than one user at the same time, by our bidding algorithm.

3.2. A Detailed Description of the Uniform Bidding Algorithm

a. *Detailed Description of variables and Constants Used.* In the following, we set

$$\beta = 1 + \frac{r_{max}}{r_{min}} \quad , \quad c = (2(2+\mu)+1)\left(\frac{r_{max}}{r_{min}}\right)^2$$

$$\lambda_2 = \mu \quad , \quad \lambda_1 = 2\,\frac{r_{max}}{r_{min}} \quad .$$

The users use the following flags: For user i, the flag $W_{ij} = 1$ iff i is a winner and is willing to get resource $\rho(j)$. The flag $A_{ij} = 1$ iff user i *accepts* the allocation of the resource $\rho(j)$. Both flags are 0 otherwise. The flag N_{ij} is initially 0, it becomes 1 when user i *releases* resource j.

The resource allocators j use the following flags: $E_{ji} = 0$ if the resource is denied and 1 if j agrees that its resource is allocated to i. Each allocator j has also a *shared* (for its parallel subprocesses) variable M_j which allows concurrent reads, and, in case of multiple writes of the same value, their sum modulo 3 is recorded. This can be done in constant (3 steps) parallel time by using the concurrent read-exclusive write model and a summation binary tree of depth 3. M_j is used to cound winners during the monitoring period.

Each user i uses also a *shared* (for all its parallel subprocesses) variable L_i. It allows concurrent reads and concurrent writes of the same value. L_i is used to identify situations in which all wanted resources have been proposed to be allocated to user i, at the same time.

The counters counter_i, counter_j count steps of respectively i,j in a round. We assume that these counters are set to zero at the beginning of each round. Note also that every time a user (or resource allocator) p (1) modifies a flag and then (2) reads a flag of a resource allocator (or user) q to see its answer, we allow for $\lambda_1 = (r_{max}/r_{min})\cdot 2$ steps between the two actions of p (these steps allow for at least 2 steps of process q so that q can read the asking flag and answer back).

We now present formally the rounds of a user i and a resource allocator j. Note that, in the code which follows, the section of code between cobegin, and coend is executed (is a synchronous fashion) by all the parallel subprocesses of the process to which the cobegin-coend block belongs.

Also note that although the semantics of "do x do-ops" and "wait until $\text{counter}_i = y$" are essentially similar, the first expression allows for an explicit wait of x steps while the second states that the process must wait until the value of the counter becomes y. We sacrifice uniformity in the code in favor of expressibility.

b. *The User's Round for User* i. (Initially $W_{ij} = A_{ij} = N_{ij} = 0$ for all $j = 1,...,k$ and $L_i = 1$)

start round

$L_i \leftarrow 1$

choose x randomly uniformly from $\{1,2,...,\beta kv\}$

if $x \neq \beta kv$ then begin do $c-1$ no-ops; go to finish end

repeat: cobegin { comment in parallel for $j = 1,...,k$ }

$W_{ij} \leftarrow 1$; do λ_1 no-ops; if $E_{ji} = 0$ then $L_i \leftarrow 0$

coend

if $L_i = 1$ AND $\text{counter}_i < c-\mu$ then

begin

cobegin { comment All resources allocated}

$A_{ij} \leftarrow 1$ { comment accept}

use resource $\rho(j)$ for μ steps

$N_{ij} \leftarrow 1$ { comment release resource}

coend

$N_{ij} \leftarrow 0$; $A_{ij} \leftarrow 0$; wait until $\text{counter}_i = c$; go to finish

end

else

begin

cobegin $A_{ij} \leftarrow 0$ { comment deny allocation} coend

if $\text{counter}_i < c-\mu-2$ then go to repeat else wait until $\text{counter}_i = c$

end

finish: end round

c. *The Resource Allocator's Round for Allocator* i. (Initially $M_j = 0$, $E_{ji} = 0$ $a_{ij} = 0$)

```
local a_ij
start round
begin
    cobegin
        do until ( counter_j = λ_2 or M_j > 0)
        begin if W_ij = 1 then (M_j ←(M_j+1)mod 3; a_ij ←1) end
        if counter_j < λ_2 then ( do no-op until counter_j = λ_2)
    coend
    cobegin
        if (M_j = 0 or M_j = 2) then
            begin E_ji ←0; go to finish end
        else
            begin
                if a_ij = 1 then
                    begin
                        E_ji ←1; wait for λ_1 steps
                        if A_ij = 0 then begin (E_ji ←0; go to finish ) end
                        else
                            begin
                                do no-op until N_ij = 1
                                { comment: Resource allocated. Await release by
                                user}
                                E_ji ←0
```

go to finish

end

end

else (repeat $E_{ji} \leftarrow 0$ until resource deallocated)

end

coend

finish: end round

3.3. Properties of the Uniform Bidding Algorithm

In the following we assume all processes tame.

PROPOSITION 1. *If a particular user i is a winner (i.e. selects $x = \beta kv$) in its current round, then its k parallel synchronous subprocesses will, at least once in its current round, report at the same time that i is a winner when all requested resources are in a monitoring period.*

Proof. Assume that user i has just been declared a winner in its current round because i got as an outcome $x = \beta kv$ in the probabilistic selection. Within one of its steps (at most r_{max}/r_{min} of a resource allocator's steps) all resource allocators are notified. If some resource allocator is in an allocating period at the time of notification, then it is going to enter a monitoring period by at most a number of is steps equal to the allocating period. If a resource allocator was in a monitoring period at the time of notification, then it shall continue being in such a period (since his resource cannot be allocated to another winner, due to the presence of winner i). So, by at most a number of steps equal to an allocating period from the time i decided that i is a winner (i.e., by $\leq (2+\mu)(r_{max}/r_{min})^2$ steps of i), all i's resources are going to be notified, at the same time, within their monitoring period, that i is a winner. □

PROPOSITION 2. *Given that a particular user i is a winner in its current round, the probability that i stays a unique winner for all its wanted resources during the whole round, is lower bounded by a constant, independent of k or v.*

Proof. We need the following definition:

DEFINITION. Let a *draw* by a user be a random independent selection of one of the numbers $\{1,2,...,\beta kv\}$.

First we prove Lemma 1.

LEMMA 1. *During a round of user i, any other distinct user j, competing for the same resource cannot draw for more than $\beta = r_{max}/r_{min} + 1$ times.*

Proof of Lemma 1. The maximum number of rounds of user j, overlapping with the round of user i, is $r_{max}/r_{min} + 1$ (because the maximum ratio of speeds cannot exceed r_{max}/r_{min} and j may have drawn at most once in each round). The "plus 1" is to take into account the fact that a round of user j may partially overlap the beginning of the round of user i. □

Now, given that user i is a winner, the probability that i remains a *unique* winner during his current round is equal to the probability that none of the competing users manages to be a winner within i's round. The number of the competing users is at most kv (at most v competitors per each of the k resources asked by user i) and each competing user can draw for at most β times within i's round (by Lemma 1). The probability of each draw failing to win is $1 - 1/\beta kv$, hence the probability that i stays a unique winner during his current round is at least

$$\left(1 - \frac{1}{\beta kv}\right)^{\beta kv} > \frac{1}{2e} \quad , \quad \text{where} \quad e = 2.73... \quad . \quad \square$$

THEOREM 3. *The probability that a user is allocated all his wanted resources in its current round is upper bounded by* $1/\beta kv$ *and lower bounded by* $1/2e\beta kv$, $e = 2.73...$

Proof. The probability $p(\Gamma_t, A)$, for oracle $\mathbf{A}$ and history Γ_t, that a user i is allocated all its wanted resources is his current round starting at t, never exceeds $1/\beta kv$ due to the fact that prob{user i chooses to be a winner in his current round} $= 1/\beta kv$. Given that user i chooses to be a winner, if i remains a unique winner during all of his current round, i is going to be allocated all of his resources (due to Proposition 1) with certainty. Multiplying probabilities given by Proposition 2, we get that

$$p(\Gamma_t, A) \geq \frac{1}{\beta kv} \frac{1}{2e} = \frac{1}{2e\beta vk} \quad .$$

THEOREM 4. *Our uniform bidding algorithm has* ϵ*-response* $O(kv \log(1/\epsilon))$ *and mean response* $O(kv)$.

Proof. Let u be the number of rounds required for user i to be granted all its k resources in some round, given that i starts requesting them at time t_1 and also assuming any history of the system up to t_1 and any oracle $\mathbf{A}$. Let round j start at time t_j, $j \leq m$. We have by Bayes' formula

$$\text{Prob}(u=m) = (1 - p(\Gamma_{t_1}, A))...(1 - p(\Gamma_{t_{m-1}}, A)) \cdot p(\Gamma_{t_m}, A) \quad .$$

By use of Theorem 3, we get

$$\text{Prob}(u=m) \leq \left(1 - \frac{1}{2e\beta kv}\right)^{m-1} \frac{1}{\beta kv} \quad .$$

If $u(\epsilon)$ is he least number such that $\text{Prob}\{u > u(\epsilon)\} \leq \epsilon$, then

Remark

$$u(\epsilon) \leq 2e\beta vk \log(\frac{2e}{\epsilon}) \quad .$$

Proof of Remark. Let $f = 1/(2e\beta kv)$ and $s = 1/(\beta kv)$. We have

$$\text{Prob}\{u = m\} \leq (1-f)^{m-1} \cdot s \quad .$$

So

$$\begin{aligned}
\text{Prob}\{u > u(\epsilon)\} &\leq \sum_{m=u(\epsilon)+1}^{\infty} (1-f)^{m-1} s \\
&\leq s(1-f)^{u(\epsilon)} \sum_{x=0}^{\infty} (1-f)^x \\
&\leq \frac{s}{f} (1-f)^{u(\epsilon)} \quad . \quad (IN\ 1)
\end{aligned}$$

It is enough for $(s/f)(1-f)^{u(\epsilon)}$ to be less than or equal to ϵ, because, then, IN1 implies $\text{Prob}\{u > u(\epsilon)\} \leq \epsilon$. So, we have

$$\begin{aligned}
\frac{s}{f}(1-f)^{u(\epsilon)} &\leq \epsilon \\
(1-f)^{u(\epsilon)} &\leq \frac{\epsilon f}{s}
\end{aligned}$$

i.e.

$$u(\epsilon) \leq \frac{\log(\frac{\epsilon f}{s})}{\log(1-f)} \quad .$$

But $f/s = 1/2e$ and $\log(1-f) > -1/f$ for $0 < f < 1$. So we get

$$u(\epsilon) \leq \frac{1}{f} \log(\frac{2e}{\epsilon})$$

i.e.

$$u(\epsilon) \leq 2e\beta kv \cdot \log(\frac{2e}{\epsilon}) \quad . \quad \square$$

To complete the proof of the theorem, let us recall that each allocation part of a resource's round takes $2+\mu$ steps. So, it is enough for the user's round to be equal to $c = (2(2+\mu)+1)(r_{max}/r_{min})^2$, by the proof of Proposition 1. This implies that the duration of a user's round is at mos cr_{max} and so (for u independent of k,v) $\text{Prob}\{\gamma_{A,k} \leq cr_{max} \cdot u(\epsilon)\} \geq 1 - \epsilon$. Hence, $\gamma_k(\epsilon) = 0(kv \log(1/\epsilon))$.

Note. Theorems 3 and 4 imply, with probability 1, that our algorithms never deadlock, no process starves, and our algorithm is probabilistically fiar, in the sense that each willing user, gets its resources infinitely often in an infinite time interval, with probability 1.

4. THE PRIORITY BIDDING ALGORITHM

4.1. Motivation

Theorems 1 and 2 provided lower bounds for systems where all the users were requesting the maximum number of resources possible. In practice, it may be the case for some users to have only a small number of requests. It is then desirable to have an implementation which tends to favor users requesting only a few resources, while, at the same time, it does not degrade a lot the performance of users requesting many resources. This section provides such an algorithm. Let us assume that at all times $t \geq 0$, $k_{i,t}$ (see section 1.2) is upper bounded by a constant k_i (and that the k_i are not necessarily the same for different users i). Our *priority bidding* algorithm has then the following property: For users i with demands k_i in $[\lfloor h_m/2 \rfloor, h_m]$ with $h_m = k \cdot 2^{-m}$, the ϵ-response is $O(h_m v \cdot \log k \cdot \log(h_m v/\epsilon))$ and the mean response is $\bar{\gamma}_{h_m} = O(h_m v \log k \log(h_m v))$. For example, if h_m is a small constant, much smaller than k and v, the mean response of users, with k_i as above, is $O(v \log k \log v)$, which is much smaller than $O(kv)$ in most cases. If, on the other hand, h_m is $\theta(k)$, then the mean response becomes $O(kv \cdot \log k \log(kv))$ compared to $O(kv)$ of the optimal algorithm, i.e. users with heavy demands suffer only a polylogarithmic degradation of their mean response time.

In our priority bidding algorithm, we split the users into groups according to their demands in number of resources. Only users of the same group compete against each other. We use the additional trick of having users execue random waits at the beginning of each round. These random waits allow user groups to avoid interfering with each other. The random waits also make the probabilistic analysis easier, by making some events probabilistically independent.

4.2. Description of the Priority Bidding Algorithm

a. *Round for User* i. The round starts with the user waiting for a randomly chosen number of steps, uniform in an interval upper bounded by a constant $c_1 = (2(2+\mu)+1)(r_{max}/r_{min})^3$ steps. Note that c_1 is chosen in such a way that $c_1 \cdot r_{min}$ is greater than the maximum possible duration of the useful part of the round. The rest of the round is the same as in our uniform bidding algorihm.

b. *Round for Resource Allocator* j. Each round of process j is split into a sequence of $\lceil \log k \rceil$ intervals. For each $m = 0,...,\lceil \log k \rceil$, in each interval Δ_m only the users i for which $k_i \in [k/2^{m+1}\rfloor, \lceil k/2^m \rceil]$ are monitored. Process j proceeds to the next interval Δ_{m+1} only if all user processes which demand k_i resources, $k_i \in [k/2^{m+1}\rfloor, \lceil k/2^m \rceil]$ have been allocated their resources. Within each Δ_m, the resource allocator goes through a sequence of "small rounds", each small round being exactly as a round of a resource allocator in our uniform bidding algorithm of Section 3.

4.3. Probabilistic Analysis of the Priority Bidding Algorithm

Let us consider a time interval Δ_m. Let u' be the number of rounds required for user i with $k_i \in [k/2^{m+1}\rfloor, \lceil k/2^m \rceil]$, and also for all users competing with user i, to have all resources allocated. The number of those users is $\leq k/2^m \cdot v$, because user i competes for at most $k/2^m$ resources, and each such resource is claimed by at most v users competing with user i. Let u be the number of rounds (within Δ_m), required just for user i. Let $u(\epsilon')$ be such that $\text{Prob}\{u \leq u(\epsilon')\} > 1 - \epsilon'$. Then, $\text{Prob}\{u' \leq +u(\epsilon')\} \geq (1-\epsilon')^{kv/2^m}$, because users take independent actions and due to the random waits. Let us set $\epsilon' = (\epsilon/kv)\cdot 2^m$. Then

$$\text{Prob}\left\{u' \leq u\left(\frac{\epsilon \cdot 2^m}{kv}\right) \geq \left(1 - \frac{\epsilon \cdot 2^m}{kv}\right)^{\frac{kv}{2^m}}\right\} \geq 1 - \epsilon \quad .$$

But, from the analysis of the uniform bidding algorithm (Theorem 3) and from the fact that, within the Δ_m, the algorithm looks exactly like the uniform bidding algorithm, we

get

$$u\left(\frac{\epsilon 2^m}{kv}\right) = 0\left(\frac{k}{2^m}\, v \log\left(\frac{kv}{2^m}\cdot\frac{1}{\epsilon}\right)\right)$$

leading to a mean response of $0((kv/2^m)\log(kv/2^m))$ for the interval Δ_m. This determines probabilistic upper bounds on the length of the interval Δ_m. Since there are $\lceil \log k \rceil$ such intervals, the ϵ-response for a user with $k_i \in [\lfloor k/2^{m+1} \rfloor, \lfloor k/2^m \rfloor]$ will be

$$0\left(\frac{kv}{2^m} \log\left(\frac{kv}{\epsilon\cdot 2^m}\right) \log k\right)$$

implying a mean response of

$$0\left(\frac{kv}{2^m} \log k \log\left(\frac{kv}{2^m}\right)\right)$$

For users with small demands (i.e., when $k/2^m = 0(k)$) the above mean response is better than the mean response of the uniform bidding algorithm of Section 3. E.g., for $2^m = \theta(k)$ we get a mean response $0(v \log k \log v)$. We hence conclude:

THEOREM 5. *Our priority bidding algorithm has the following property: For users i with demands k_i in $[\lfloor h_m/2 \rfloor, h_m]$ and $h_m = k\cdot 2^{-m}$, the ϵ-response is*

$$0\left(h_m v \log k \log\left(\frac{h_m v}{\epsilon}\right)\right)$$

and the mean response is $\bar{\gamma}_{k_m} = 0(h_m v \log k \log(h_m v))$.

□

REFERENCES

Andrews, G., "Synchronizing Resources", *ACM Trans. on Programming Languages and Systems* **3** (4), 405-30 (1981).

Angluin, D., "Local and Global Properties in Networks of Processors", *12th Annual Symp. on Theory of Computing,* Los Angeles, CA, 82-93 (April 1980).

Arjomandi, E., M. Fischer, and N. Lynch, "A Difference in Efficiency Between Synchronous and Asynchronous Systems", *13th Ann. Symp. on Theory of Computing,* (April 1981).

Bernstein, A.J., "Output Guards and Nondeterminism in Communicating Sequential Processes", *ACM Trans. on Programming Languages and Systems* **2** (2), 234-238 (1980).

Bernstein, P. and N. Goodman, "Fundamental Algorithms for Concurrency Control in Distributed Database Systems", CCA Tr. Contract No. F30603-79-0191, Cambridge, MA (1980).

Dennis, J.B. and D.P. Misunas, "A Preliminary Architecture for a Basic Dataflow Processor", *Proc. 2nd Annual Symp. on Computer Archiecture, ACM IEEE,* 126-132 (1974).

Fisher, M.J., N.A. Lynch, J.E. Burns, and A. Borodin, "Resource Allocation with Immunity to Limited Process Failure", *19th FOCS,* 234-254 (1979).

Francez, N., "Extended Naming Conventions for Communicating Processes", *9th ACM Symp. on Principles of Programming Languages,* Albuquerque, New Mexico, (Jan. 1982).

Francez, N. and J. Reif, "A Social CSP", to appear.

Francez, N. and M. Rodeh, "A Distributed Data Type Implemented by a Probabilistic Communication Scheme", *21st Ann. Symp. on Foundations of Computer Sci-*

ence, Syracuse, New York, 373-379 (Oct. 1980).

Hart, S. and M. Sharir, "Termination of Probabilistic Concurrent Programs", *9th Ann. SCM Symp. on Principles of Programming Languages,* Albuquerque, New Mexico, (Jan. 1982).

Hoare, C.A.R., "Communicating Sequential Processes", *Com. of ACM,* **21** (8), 666-677 (1978).

Itai, A. and M. Rodeh, "Symmetry Breaking in Distributive Networks", *22nd Annual Symp. on Foundations of Computer Science,* Nashville, Tennessee, 120-158 (Oct. 1981).

Lehmann, D. and M. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers' Problem", in *8th ACM Symp. on Principles of Programming Languages,* (Jan. 1981).

Lipton, R. and F.G. Sayward, "Response Time of Parallel Programs", Research Report #108, Dept. of Comp. Science, Yale University, (June 1977).

Lynch, M.A., "Fast Allocation of Nearby Resources in a Distributed System", *12th Ann. Symp. in Theory of Computing,* Los Angeles, CA, 70-81 (April 1980).

Rabin, M., "N-Process Synchronization by a $4 \log M$- Valued Shared Variable", *21st Ann. Symp. on Foundations of Comp. Science,* Syracuse, New York, 407-410 (Oct. 1980).

Rabin, M., "The Choice Coordination Problem", Mem. No. UCB/ERL MBO/38, Electric. Research Laboratory, University of California, Berkeley, (Aug. 1980).

Reif, J.H. and P. Spirakis, "Distributed Algorithms for Synchronizing Interprocess Communication Within Real Time", *13th Ann. Symp. on Theory of Computation,* Wisconsin, 133-145 (1981); also as "Real-Time Synchronization of Interprocess Communications", in *ACM Transactions on Programming Languages and Systems,* **6,** No. 4, April, 1984, pp. 215-238.

Reif, J.H. and P. Spirakis, "Unbounded Speed Variability in Distributed Communications Systems", *9th ACM Symp. on Principles of Programming Languages,* Albuquerque, New Mexico, (1982a); also in *SIAM Journal of Computing,* **14,** No. 1, February, 1985, pp. 75-92.

Reif, J.H. and P. Spirakis, "Real Time Resource Allocation in Distributed Systems", *ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing,* Ottawa, Canada, (Aug. 1982b).

Schwarz, J., "Distributed Synchronization of Communicating Sequential Processes", DAI Research Report No. 56, University of Edinburg, (1980).