A Family of Distributed Deadlock Avoidance Protocols and Their Reachable State Spaces*

César Sánchez, Henny B. Sipma, and Zohar Manna

Computer Science Department
Stanford University, Stanford, CA 94305-9025
{cesar,sipma,manna}@CS.Stanford.EDU

Abstract. We study resource management in distributed systems. Incorrect handling of resources may lead to deadlocks, missed deadlines, priority inversions, and other forms of incorrect behavior or degraded performance. While in centralized systems deadlock avoidance is commonly used to ensure correct and efficient resource allocation, distributed deadlock avoidance is harder, and general solutions are considered impractical due to the high communication overhead. However, solutions that use only operations on local data exist if some static information about the possible sequences of remote invocations is known.

We present a family of efficient distributed deadlock avoidance algorithms that subsumes previously known solutions as special instances. Even though different protocols within the family allow different levels of concurrency and consequently fewer or more executions, we prove that they all have the same set of reachable states, expressed by a global invariant. This result enables: (1) *a design principle*: the use of different protocols at different sites does not compromise deadlock avoidance; (2) *a proof principle*: any resource allocation protocol that preserves the global invariant and whose allocation decisions are at least as liberal as those of the least liberal in the family, guarantees absence of deadlock.

1 Introduction

Middleware services play a key role in the development of modern distributed real-time and embedded (DRE) systems. DRE systems often consist of a variety of hardware and software components, each with their own protocols, interfaces, operating systems, and API's. Middleware services hide this heterogeneity, allowing the software engineer to focus on the application, by providing a high-level uniform interface, and handling management of resources and communication and coordination between components. However, this approach is effective only if these services are well-defined, flexible and efficient.

In this paper we focus on resource allocation services for DRE systems. Computations in distributed systems often involve a distribution of method calls over

* This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363, and CCF-0430102, and by NAVY/ONR contract N00014-03-1-0939.

M.B. Dwyer and A. Lopes (Eds.): FASE 2007, LNCS 4422, pp. 155–169, 2007.

multiple sites. At each site these computations need resources, for example in the form of threads, to proceed. With multiple processes starting and running at different sites, and a limited number of threads at each site, deadlock may arise. Traditionally three methods are used to deal with deadlock: prevention, avoidance and detection. In *deadlock prevention* a deadlock state is made unreachable by, for example, imposing a total order in which resources are acquired, such as in "monotone locking" [1,4]. This strategy can substantially reduce performance, by artificially limiting concurrency. With *deadlock detection*, common in databases, deadlock states may occur, but are upon detection resolved by, for example, roll-back of transactions. In embedded systems, however, this is usually not an option, especially in systems interacting with physical devices.

Deadlock avoidance methods take a middle route. At runtime a protocol is used to decide whether a resource request is granted based on current resource availability and possible future requests of processes in the system. A resource is granted only if it is *safe*, that is, if there is a strategy to ensure that all processes can complete. To make this test feasible, processes that enter the system must announce their possible resource usage. This idea was first proposed in centralized systems by Dijkstra in his Banker's algorithm [2,3,13,11], where processes report the maximum number of resources that they can request. When resources are distributed across multiple sites, however, deadlock avoidance is harder because the different sites may have to consult each other to determine whether a particular allocation is safe. Consequently, a general solution to distributed deadlock avoidance is considered impractical [12]; the communication costs involved simply outweigh the benefits gained from deadlock avoidance over deadlock prevention.

We study distributed deadlock avoidance algorithms that do not require any communication between sites. Our algorithms are applicable to distributed systems in which processes perform remote method invocations and lock local resources (threads) until all remote calls have returned. In particular, if the chain of remote calls arrives back to a site previously visited, then a new resource is needed. This arises, for example, in DRE architectures that use the *WaitOnConnection* policy for nested up-calls [9,10,14]. Our algorithms succeed in providing deadlock avoidance without any communication overhead by using static process information in the form of call graphs that represent all possible sequences of remote invocations. In DRE systems, this information can usually be extracted from the component specifications or from the source code directly by static analysis.

In this paper we analyze the common properties of a family of deadlock avoidance protocols that include the protocols we presented in earlier papers [8,7,6]. We show that the two protocols Basic-P introduced in [8] and Live-P presented in [6] are the two extremes of a spectrum of protocols that allow, going from Basic-P to Live-P, increasing levels of concurrency. Despite these different levels of concurrency, and thus executions permitted, we prove that all protocols in the family have the same set of reachable states. The significance of this result is that it allows running different protocols from the family at different sites without compromising deadlock. In addition, it considerably simplifies proving

correct modifications and refinements of these protocols, as proofs reduce to showing that the new protocol preserves the same invariant.

The rest of this paper is structured as follows. Section 2 describes the computational model and Section 3 introduces the protocols. Section 4 characterizes the different levels of concurrency by comparing allocation sequences for the different protocols, and Section 5 presents the proof that all protocols have the same set of reachable states. Section 6 concludes with some remarks about the design principle enabled by our results and some open problems.

2 Computational Model

We model a distributed system as a set of sites that perform computations and a call graph, which provides a static representation of all possible resource usage patterns. Formally, a distributed system is a tuple $\mathcal{S} : \langle \mathcal{R}, \mathcal{G} \rangle$ consisting of

- $\mathcal{R} : \{r_1, \ldots, r_{|\mathcal{R}|}\}$, a set of sites, and
- $\mathcal{G} : \langle V, \rightarrow, I \rangle$ a call graph specification.

A call graph specification $\mathcal{G} : \langle V, \rightarrow, I \rangle$ consists of a directed acyclic graph $\langle V, \rightarrow \rangle$, which captures all the possible sequences of remote calls that processes can perform. The set of initial nodes $I \subseteq V$ contains those methods that can be invoked when a process is spawned. A call-graph node $n{:}r$ represents a method n that runs in site r. We also say that node n *resides* in site r. If two nodes reside in the same site we write $n \equiv_{\mathcal{R}} m$. An edge from $n{:}r$ to $m{:}s$ denotes that method n, in the course of its execution may invoke method m in site s.

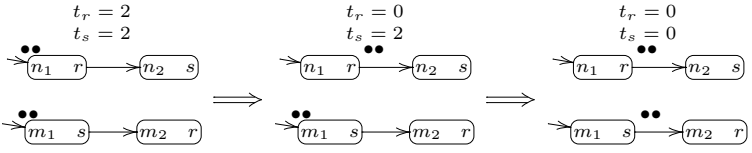
We assume that each site has a fixed number of pre-allocated resources. Although in many modern operating systems threads can be spawned dynamically, many DRE systems pre-allocate fixed sets of threads to avoid the relatively large and variable cost of thread creation and initialization. Each site r maintains a set of local variables V_r that includes the constant $T_r \geq 1$ denoting the number of resources present in r, and a variable t_r that represents the number of available resources. Initially, $t_r = T_r$.

The execution of a system consists of processes, created dynamically, executing computations that only perform remote calls according to the edges in the call graph. When a new process is spawned it starts its execution with the graph node whose outgoing paths describe the remote calls that the process can perform. All invocations to a call graph node require a new resource in the corresponding site, while call returns release a resource. We impose no restriction on the topology of the call graph or on the number of process instances, and thus deadlocks can be reached if all requests for resources are immediately granted.

Example 1. Consider a system with two sites $\mathcal{R} = \{r, s\}$, a call graph with four nodes $V = \{n_1, n_2, m_1, m_2\}$, where n_1 and m_1 are initial, and edges:

This system has reachable deadlocks if no controller is used. Let sites s and r handle exactly two threads each. If four processes are spawned, two instances of

n_1 and two of m_1, all resources in the system will be locked after each process starts executing its initial node. Consequently, the allocation attempts for n_2 and m_2 will be blocked indefinitely, so no process will terminate or return a resource. This allocation sequence is depicted below, where a • represents an existing process that tries to acquire a resource at a node (if • precedes the node) or has just been granted the resource (if • appears after the node).



□

Our deadlock avoidance solution consists of two parts: (1) the offline calculation of *annotations*, maps from call-graph nodes to natural numbers; and (2) a run-time *protocol* that controls resource allocations based on these annotations. Informally, an annotation measures the number of resources required for a computation. The protocols grant a request based on the remaining local resources (and possibly other local variables) and the annotation of the requesting node.

Protocol. A *protocol* for controlling the resource allocation in node $n : r$ is implemented by a program executed in r before and after method n is dispatched. This code can be different for different call-graph nodes even if they reside in the same site. The schematic structure of a protocol for a node $n{:}r$ is:

$$n :: \left[\begin{array}{ll} \boxed{\textbf{when } En_n(V_r) \textbf{ do } In_n(V_r, V_r')} & \left.\vphantom{\boxed{x}}\right\} \textit{entry section} \\ f() & \left.\vphantom{\boxed{x}}\right\} \textit{method invocation} \\ \boxed{Out_n(V_r, V_r')} & \left.\vphantom{\boxed{x}}\right\} \textit{exit section} \end{array}\right]$$

Upon invocation, the entry section checks resource availability by inspecting local variables V_r of site r. If the predicate $En_n(V_r)$, called the enabling condition, is satisfied we say that the entry section is *enabled*. In this case, the request can be granted and the local variables are updated according to the relation $In_n(V_r, V_r')$ (where V_r' stands for the local variables after the action is taken). We assume that the entry section is executed atomically, as a *test-and-set*. The method invocation section executes the code of the method, here represented as $f()$, which may perform remote calls according to the edges outgoing from node n in the call graph. The method invocation can only terminate after all its invoked calls (descendants in the call graph) have terminated and returned. The exit section releases the resource and may update some local variables in site r, according to the relation $Out_n(V_r, V_r')$. In_n is called the entry action and Out_n is called the exit action.

Annotations. Given a system $\mathcal{S}$ and annotation α, the annotated call graph $(V, \rightarrow, \dashrightarrow)$ is obtained from the call graph $(V, \rightarrow)$ by adding one edge $n \dashrightarrow m$ between any two nodes that reside in the same site with annotation $\alpha(n) \geq \alpha(m)$. A node n "depends" on a node m, which we represent $n \succ m$, if there is a path in the annotated graph from n to m that follows at least one $\rightarrow$ edge. The annotated graph is acyclic if no node depends on itself, in which case we say that the annotation is acyclic.

3 A Family of Local Protocols

Our goal is to construct protocols that (1) avoid deadlock in all scenarios, (2) require no communication between sites, and (3) maximize resource utilization (grant requests as much as possible without compromising deadlock freedom).

The first protocol we proposed was Basic-P [8], shown in Fig. 1 for a node $n{:}r$ with annotation $\alpha(n) = i$. Upon a resource request, Basic-P checks whether the number of available resources is large enough, as indicated by the annotation i. This check ensures that processes (local or remote) that could potentially be blocked if the resource is granted, have enough resources to complete. The correctness of Basic-P is based on the acyclicity of the annotations:

$$n :: \left[\begin{array}{l} \left[\begin{array}{c}\textbf{when } i \leq t_r \textbf{ do} \\ t_r\text{--}\end{array}\right] \\ f() \\ t_r\text{++}\end{array}\right]$$

Fig. 1. The protocol Basic-P

Theorem 1 (Annotation Theorem for Basic-P [8]). *Given a system $\mathcal{S}$ and an acyclic annotation, if* Basic-P *is used to control resource allocations then all executions of $\mathcal{S}$ are deadlock free.*

Example 2. Reconsider the system from Example 1. The left diagram below shows an annotated call graph with $\alpha(n_1) = \alpha(n_2) = \alpha(m_2) = 1$ and $\alpha(m_1) = 2$. It is acyclic, and thus by Theorem 1, if Basic-P is used with these annotations, the system is deadlock free.

Let us compare this with Example 1 where a resource is granted simply if it is available. This corresponds to using Basic-P with the annotated call graph above on the right, with $\alpha(n) = 1$ for all nodes. In Example 1 we showed that a deadlock is reachable, and indeed this annotated graph is not acyclic; it contains dependency cycles, for example $n_1 \rightarrow n_2 \dashrightarrow m_1 \rightarrow m_2 \dashrightarrow n_1$. Therefore Theorem 1 does not apply. In the diagram on the left all dependency cycles are broken by the annotation $\alpha(m_1) = 2$. Requiring the presence of at least two resources for granting a resource at m_1 ensures that the last resource available in s can only be obtained at n_2, which breaks all possible circular waits. □

$$n :: \left[\begin{array}{l} \left[\begin{array}{c}\textbf{when } 1 \le t_r \textbf{ do}\\ t_r\text{--}\end{array}\right] \\ f() \\ t_r\text{++}\end{array}\right] \qquad n :: \left[\begin{array}{l} \left[\begin{array}{c}\textbf{when } i \le p_r \ \wedge\ 1 \le t_r \textbf{ do}\\ \langle p_r\text{--},\ t_r\text{--}\rangle\end{array}\right] \\ f() \\ \langle t_r\text{++},\ p_r\text{++}\rangle\end{array}\right]$$

If $i = 1$ If $i > 1$

Fig. 2. The protocol Efficient-P

The protocol Basic-P can be improved using the observation that processes requesting resources to execute nodes with annotation 1 can always terminate, in spite of any other process in the system. This observation leads to Efficient-P (shown in Fig. 2) , which uses two counters: t_r, as before; and p_r, to keep track of the "potentially recoverable" resources, which include not only the available resources but also the resources granted to processes in nodes with annotation 1. A similar version of the Annotation Theorem for Efficient-P establishes that in the absence of dependency cycles, Efficient-P can reach no deadlocks.

The proof of the Annotation Theorem for Basic-P and Efficient-P [8] relies on showing that the following global invariant φ is maintained:

$$\varphi \stackrel{\text{def}}{=} \bigwedge_{r\in\mathcal{R}} \bigwedge_{k\le T_r} \varphi_r[k] \qquad \text{with} \qquad \varphi_r[k] \stackrel{\text{def}}{=} A_r[k] \le T_r - (k-1).$$

where $a_r[k]$ stands for the number of active processes running in site r executing nodes with annotation k and $A_r[k]$ stands for $\sum_{j\ge k} a_r[j]$, that is, $A_r[k]$ represents the number of active processes running in site r executing nodes with annotation k or higher. In [6] we exploited this fact by constructing the protocol Live-P, shown in Fig. 3, which grants access to a resource precisely whenever φ_r is preserved. This protocol not only guarantees absence of deadlock, it also provides, in contrast with Basic-P, individual liveness for all processes. Its enabling condition, $\varphi_r^{(i)}$, is exactly the weakest precondition for φ_r of the transition that grants the resource:

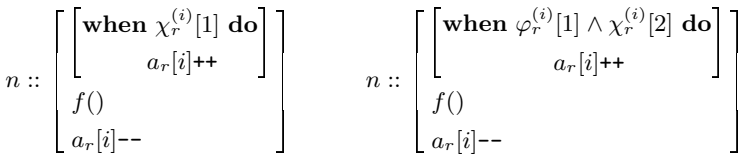
$$n :: \left[\begin{array}{l} \left[\begin{array}{c}\textbf{when } \varphi_r^{(i)} \textbf{ do}\\ a_r[i]\text{++}\end{array}\right] \\ f() \\ a_r[i]\text{--}\end{array}\right]$$

Fig. 3. The protocol Live-P

$$\varphi_r^{(i)} \stackrel{\text{def}}{=} \left(\begin{array}{ll} \bigwedge_{k>i} A_r[k] & \le T_r - (k-1) \\ & \wedge \\ \bigwedge_{k\le i} A_r[k] + 1 & \le T_r - (k-1)\end{array}\right)$$

We use $\varphi_r^{(i)}[j]$ for the clause of $\varphi_r^{(i)}$ that corresponds to annotation j. Observe that $\varphi_r^{(i)}[j]$ is syntactically identical to $\varphi_r[j]$ for $j > i$. Moreover, for $j \le i$, $\varphi_r^{(i)}[j]$ implies $\varphi_r[j]$.

To compare the protocols we restate Basic-P and Efficient-P in terms of the notation introduced for Live-P. The enabling condition of Basic-P becomes:



(a) 1-Efficient-P (b) 2-Efficient-P

Fig. 4. Basic-P and Efficient-P restated using strengthenings

$$A_r[1] \leq T_r - (i-1)$$

which is, as we will see, stronger than $\varphi_r^{(i)}$, that is, the enabling condition of Basic-P implies that of Live-P. Given $k \leq i$ we define the k-th strengthening formula for a request in node $n{:}r$ with annotation i as:

$$\chi_r^{(i)}[k] \stackrel{\text{def}}{=} A_r[k] \leq T_r - (i-1)$$

It is easy to see that the following holds for all $k \leq j \leq i$,

$$\chi_r^{(i)}[k] \rightarrow \varphi_r^{(i)}[j] \qquad \text{and therefore} \qquad \chi_r^{(i)}[k] \rightarrow \bigwedge_{k \leq j \leq i} \varphi_r^{(i)}[j].$$

Also, if φ_r holds before the resource is granted, then $\varphi_r^{(i)}[j]$ also holds for all $i \geq j$, since the formulas for $\varphi_r^{(i)}[j]$ and $\varphi_r[j]$ are identical in this case. Hence:

$$\chi_r^{(i)}[k] \rightarrow \bigwedge_{k \leq j} \varphi_r^{(i)}[j]. \tag{1}$$

Finally, if $\varphi_r^{(i)}[j]$ is satisfied for all values less than k, and $\chi_r^{(i)}[k]$ is ensured, $\varphi_r^{(i)}$ can be concluded:

$$\begin{aligned}
(\bigwedge_{j<k} \varphi_r^{(i)}[j]) \wedge \chi_r^{(i)}[k] &\rightarrow (\bigwedge_{j<k} \varphi_r^{(i)}[j]) \wedge (\bigwedge_{j \geq k} \varphi_r^{(i)}[j]) \\
&\leftrightarrow \bigwedge \varphi_r^{(i)}[j] \\
&\leftrightarrow \varphi_r^{(i)}.
\end{aligned}$$

Therefore, if a protocol ensures that for some k, both $\bigwedge_{j<k} \varphi_r^{(i)}[j]$ and the k-strengthening $\chi_r^{(i)}[k]$ hold, then φ_r is an invariant.

In general, the lower the value of the strengthening point k, the less computation is needed to compute the predicate (the number of comparisons is reduced) but the less liberal the enabling condition becomes. In the case of $k = 1$ the strengthening is $\chi_r^{(i)}[1]$, and the protocol obtained (see Fig. 4(a)) is equivalent

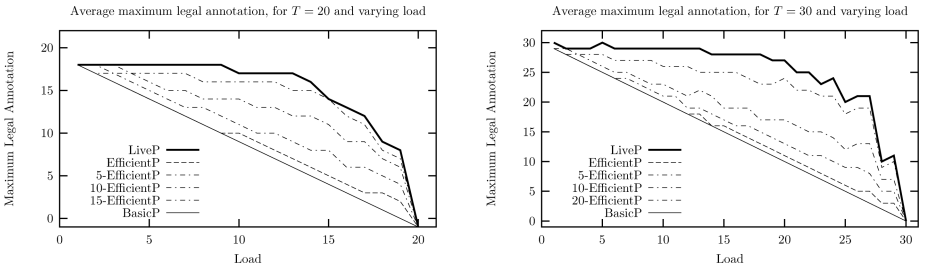


Fig. 5. A comparison of BASIC-P, k-EFFICIENT-P and LIVE-P

to BASIC-P. Note that this protocol is *logically* equivalent to BASIC-P: the result of the enabling condition, and the effect of the input and output actions on future tests are the same. The implementation of BASIC-P introduced earlier uses a single counter t_r, while in this restated version, several counters are used: $a_r[i]$ and $A_r[1]$. However, the effect on $A_r[1]$ of the increments and decrements of $a_r[i]$ are independent of i. Therefore, these actions can be implemented as $A_r[1]$**++** and $A_r[1]$**--** respectively. Similarly, with a strengthening point of $k = 2$ we obtain a protocol equivalent to EFFICIENT-P, shown in Fig. 4(b).

The general form of our family of protocols can now be given as k-EFFICIENT-P, shown in Fig. 6. It covers the full spectrum of protocols with BASIC-P, which is equivalent to 1-EFFICIENT-P, at one end and LIVE-P, which is equal to T_r-EFFICIENT-P at the other end of the spectrum. The protocols k-EFFICIENT-P can be implemented in several ways. The simplest implementation needs space $O(k \ log \ T_r)$ to store k counters and requires $O(k)$ operations per allocation decision. A more sophisticated implementation using an *active tree* data-structure still needs $O(k \ log \ T_r)$, but requires only $O(log \ k)$ operations per allocation decision [6]. Fig. 5 presents some experimental results that compare concurrency levels allowed by the different protocols. The figures depict the maximum annotation allowed by each protocol as a function of the load (total number of active processes). The load is created by annotations picked uniformly at random.

$$n :: \left[\begin{array}{l} \left[\begin{array}{l} \textbf{when } \big(\bigwedge_{j<k} \varphi_r^{(i)}[j]\big) \wedge \chi_r^{(i)}[k] \textbf{ do} \\ \qquad a_r[i]\texttt{++} \end{array}\right] \\ f() \\ a_r[i]\texttt{--} \end{array}\right]$$

Fig. 6. The protocol k-EFFICIENT-P

4 Allocation Sequences

In this section we compare the set of runs allowed by each protocol. We capture these sets by languages over an alphabet of allocations and deallocations.

Given a call graph $(V, \rightarrow)$ let the set $\overline{V}$ contain a symbol $\overline{n}$ for every n in V. The allocation alphabet Σ is the disjoint union of V and $\overline{V}$. Symbols in V

are called allocation symbols, while symbols in $\overline{V}$ are referred to as deallocation symbols. Given a string s in Σ^* and a symbol v in Σ we use s_v for the number of occurrences of v in s, and $|s|_n$ to stand for $s_n - s_{\overline{n}}$. A *well-formed* allocation string s is one for which every deallocation occurs after a matching allocation, that is, for every prefix p of s, $|p|_n \geq 0$. An *admissible* allocation sequence is one that corresponds to a prefix run of the system, according to the call graph. This requires (1) that the string be well-formed, (2) that every allocation of a non-root node is preceded by a matching allocation of its parent node, and (3) that every deallocation of a node is preceded by a corresponding deallocation of its children nodes. Formally,

Definition 1 (Admissible Strings). *A well-formed allocation string s is called admissible if for every prefix p of s, and every remote call $n \to m$: $|p|_n \geq |p|_m$.*

Admissible strings ensure that the number of child processes (callees) is not higher than the number of parents (caller processes), so that there is a possible match. For brevity, we simply use *string* to refer to admissible string.

We say that a protocol is *completely local* if all the enabling conditions are determined by: (1) the annotation of the call-graph node requested, and (2) the set of active processes in the local site and their annotations. It is easy to see that the protocols k-Efficient-P are completely local. We use the values of $a_r[\cdot]$ and $A_r[\cdot]$ as the (abstract) global states of the system since these values capture all effects of completely local protocols in the outcomes of future requests. The initial state of the system, denoted by Θ, is $a_r[i] = A_r[i] = 0$ for all sites r and annotations i.

Given a state σ and a protocol P, if the enabling condition of P for a node n is satisfied at σ we write $En_n^P(\sigma)$. For convenience, we introduce a new state $\perp$ to capture sequences that a protocol forbids, and require that $En_n^P(\perp)$ does not hold. We denote by $P(s)$ the[1] state reached by P after exercising the allocation string s, defined inductively as $P(\epsilon) = \Theta$ and:

$$P(s\ n) = \begin{cases} In_n^P(P(s)) & \text{if } En_n^P(P(s)) \\ \perp & \text{otherwise} \end{cases} \qquad P(s\ \overline{n}) = \begin{cases} Out_n^P(P(s)) & \text{if } P(s) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

We say that a string s is accepted by a protocol P if $P(s) \neq \perp$. The set of strings accepted by P is denoted by $\mathcal{L}(P)$, and we use $P \sqsubseteq Q$ for the partial order defined by language inclusion $\mathcal{L}(P) \subseteq \mathcal{L}(Q)$.

Example 3. Reconsider the system in Example 1. The allocation sequence that leads to a deadlock is $s : n_1 n_1 m_1 m_1$. Even though $n_1 n_1 m_1$ is in $\mathcal{L}$(Basic-P), the enabling condition of m_1 becomes disabled, so Basic-P$(s) = \perp$ and $s \notin \mathcal{L}$(Basic-P). □

[1] All our protocols are deterministic but the results can be adapted for non-deterministic protocols as well.

Lemma 1. *The following are equivalent:*

(i) $\mathcal{L}(P) \subseteq \mathcal{L}(Q)$.
(ii) For all strings s *and allocation symbols* n*, if* $En_n^P(P(s))$ *then* $En_n^Q(Q(s))$.

Proof. We prove both implications separately:

- Assume $\mathcal{L}(P) \subseteq \mathcal{L}(Q)$, and let s and n be such that $En_n^P(P(s))$. Since $s \in \mathcal{L}(P)$ then $s \in \mathcal{L}(Q)$. Moreover, $s \cdot n \in \mathcal{L}(P)$ and then $s \cdot n \in \mathcal{L}(Q)$. Hence, $En_n^Q(Q(s))$.
- Assume now (ii). We reason by induction on strings:
  - First, both $\epsilon \in \mathcal{L}(P)$ and $\epsilon \in \mathcal{L}(Q)$.
  - Let $s \cdot n \in \mathcal{L}(P)$. Then $En_n^P(P(s))$, so also $En_n^Q(Q(s))$. Hence, $s \cdot n \in \mathcal{L}(Q)$.
  - Let $s \cdot \overline{n} \in \mathcal{L}(P)$. This implies $s \in \mathcal{L}(P)$ and by inductive hypothesis $s \in \mathcal{L}(Q)$. Then $s \cdot \overline{n} \in \mathcal{L}(Q)$, as desired.

Therefore (i) and (ii) are equivalent. □

Let P, Q be any two of Basic-P, Efficient-P, k-Efficient-P and Live-P. We showed in Section 3 that the entry and exit actions are identical for all these protocols. Therefore, if s is in the language of both P and Q then the states reached are the same, i.e., $P(s) = Q(s)$. It follows that if for all states σ, $En_n^P(\sigma)$ implies $En_n^Q(\sigma)$, then $\mathcal{L}(P) \subseteq \mathcal{L}(Q)$.

Lemma 2. *If* j-Efficient-P *allows an allocation then* k-Efficient-P *also allows the allocation, provided* $j \leq k$.

Proof. Let $j \leq k$. It follows from the definition that $\chi_r^{(i)}[j]$ implies $\chi_r^{(i)}[k]$. Moreover, by (1), $\chi_r^{(i)}[j]$ implies $\bigwedge_{j \leq l \leq k} \varphi_r^{(i)}[l]$. Consequently,

$$\underbrace{\bigwedge_{l<j} \varphi_r^{(i)}[l] \wedge \chi_r^{(i)}[j]}_{En_n^{j\text{-Efficient-P}}} \rightarrow \underbrace{\bigwedge_{l<k} \varphi_r^{(i)}[l] \wedge \chi_r^{(i)}[k]}_{En_n^{k\text{-Efficient-P}}}$$

Therefore if j-Efficient-P allows a request so does k-Efficient-P. □

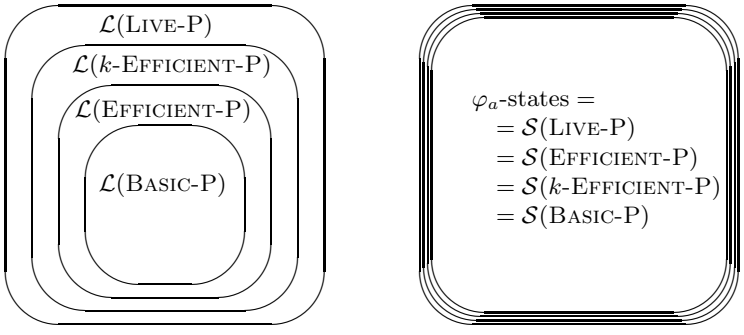
Lemma 2 states that the enabling condition of k-Efficient-P becomes weaker as k grows, that is, the enabling condition of Basic-P is stronger than that of Efficient-P, which in turn is stronger than k-Efficient-P, which is stronger than Live-P. An immediate consequence of Lemma 2 is:

$$\text{Basic-P} \sqsubseteq \text{Efficient-P} \sqsubseteq \ldots \sqsubseteq k\text{-Efficient-P} \sqsubseteq \ldots \sqsubseteq \text{Live-P}$$

The following examples show that these language containments are strict:

$$\text{Basic-P} \not\supseteq \text{Efficient-P} \not\supseteq \ldots \not\supseteq k\text{-Efficient-P} \not\supseteq \ldots \not\supseteq \text{Live-P}$$

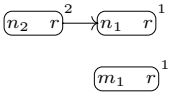
which is depicted in Fig 7(a).



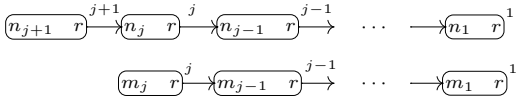
(a) Allocation sequences (b) Reachable state spaces

Fig. 7. Comparison of the family of protocols

Example 4. Consider the following call-graph, with initial resources $T_r = 2$.



The string $m_1 n_2$ is accepted by Efficient-P but not by Basic-P. This system can be generalized to show that there is a string accepted by k-Efficient-P but not by j-Efficient-P (for $j < k$). Consider the following annotated call graph, with initial resources $T_r = j + 1$.



The string $m_j n_{j+1}$ is accepted by k-Efficient-P, but is not accepted by j-Efficient-P. □

5 Reachable State Spaces

The reachable state space of a protocol P, denoted by $\mathcal{S}(P)$, is the set of global states that P can reach following some admissible allocation sequence. Clearly, for two protocols P and Q, if their actions are equivalent and $P \sqsubseteq Q$ then every state reachable by P is also reachable by Q. Indeed any allocation string that reaches a state for P also reaches that same state for Q.

Lemma 3. *For every two protocols P and Q with the same entry and exit actions, if $P \sqsubseteq Q$ then $\mathcal{S}(P) \subseteq \mathcal{S}(Q)$.*

Consequently,

$$\mathcal{S}(\text{Basic-P}) \subseteq \mathcal{S}(\text{Efficient-P}) \subseteq \ldots \subseteq \mathcal{S}(k\text{-Efficient-P}) \subseteq \ldots \subseteq \mathcal{S}(\text{Live-P})$$

Let $\mathcal{S}(\varphi_a)$ describe the set of states that satisfy φ and that are reachable by some admissible allocation string. In the rest of this section we show that the above containment relation collapses into equalities by proving

$$\mathcal{S}(\text{Basic-P}) \;=\; \mathcal{S}(\text{Live-P}) \;=\; \mathcal{S}(\varphi_a)$$

The proof relies on the existence of a *preference order* on the nodes of the annotated call graph, such that, if allocations are made following this order, then every allocation request that succeeds in Live-P also succeeds in Basic-P.

5.1 Preference Orders

A *preference order* of an annotated call graph is an order on the nodes such that, if all allocations in a given admissible string are performed following that order, then (1) the sequence obtained is also admissible, and (2) higher annotations for each site are visited first. This will allow us to show that Basic-P can reach all φ_a-states.

Given a call graph, a total order $>$ on its nodes is called topological if it respects the descendant relation, that is, if for every pair of nodes n and m, if $n \rightarrow m$ then $n > m$. Analogously, we say that an order $>$ respects an annotation α if for every pair of nodes n and m residing in the same site, if $\alpha(n) > \alpha(m)$ then $n > m$. A total order that is topological and respects annotations is called a preference order.

Lemma 4. *Every acyclically annotated call graph has a preference order.*

Proof. The proof proceeds by induction on the number of call-graph nodes. The result trivially holds for the empty call graph. For the inductive step, assume the result holds for all call graphs with at most k nodes and consider an arbitrary call graph with $k+1$ nodes.

First, there must be a root node whose annotation is the highest among all the nodes residing in the same site. Otherwise a dependency cycle can be formed: take the maximal nodes for all sites, which are internal by assumption, and their root ancestors. For every maximal (internal) node there is $\rightarrow^+$ path reaching it, starting from its corresponding root. Similarly, for every root there is an incoming $\dashrightarrow$ edge from the maximal internal node that resides in its site. A cycle exists since the (bipartite) subgraph of roots and maximal nodes is finite, and every node has a successor (a $\rightarrow^+$ for root nodes, and a $\dashrightarrow$ for maximal nodes). This contradicts that the annotation is acyclic.

Now, let n be a maximal root node, and let $>$ be a preference order for the graph that results by removing n, which exists by inductive hypothesis. We extend $>$ by adding $n > m$ for every other node m. The order is topological since n is a root. The order respects annotations since n is maximal in its site. □

5.2 Reachable States

A global state of a distributed system is *admissible* if all existing processes (active or waiting) in a node n are also existing, and active, in every node ancestor of

n. That is, if the state corresponds to the outcome of some admissible allocation sequence.

Theorem 2. *The set of reachable states of a system using* Live-P *is precisely the set of φ_a-states.*

Proof. It follows directly from the specification of Live-P that all reachable states satisfy φ. Therefore, we only need to show that all φ_a-states are reachable.

We proceed by induction on the number of active processes in the system. The base case, with no active process, is the initial state of the system Θ, which is trivially reachable by Live-P. For the inductive step, consider now an arbitrary φ_a-state σ with some active process. Since the call graph is acyclic and finite, there must be some active process P in σ with no active descendants. The state σ' obtained by removing P from σ is an admissible φ_a-state (all the conditions of admissibility and the clauses of φ are either simplified or identical); by the inductive hypothesis, σ' is reachable by Live-P. Since σ is obtained from σ' by an allocation that preserves φ (otherwise σ would not be a φ_a-state), then σ is reachable by Live-P. □

Theorem 2 states that for every sequence s that leads to a φ_a-state there is a sequence s' arriving at the same state for which all prefixes also reach φ_a-states. The sequence s' is in the language of Live-P.

Perhaps somewhat surprisingly, the set of reachable states of Basic-P is also the set of all φ_a-states. To prove this we first need an auxiliary lemma.

Lemma 5. *In every φ_a-state, an allocation request in site r with annotation k has the same outcome using* Basic-P *and* Live-P*, if there is no active process in r with annotation strictly smaller than k.*

Proof. First, in every φ_a-state, if Basic-P grants a resource so does Live-P, by Lemma 2. We need to show that in every φ_a-state, if Live-P grants a request of k and $a_r[j] = 0$ for all $j < k$, then Basic-P also grants the request. In this case,

$$T_r - t_r = A_r[1] = \sum_{j=1}^{T_r} a_r[j] = \sum_{j=k}^{T_r} a_r[j] = A_r[k], \tag{2}$$

and since Live-P grants the request, then $A_r[k] + 1 \leq T_r - (k-1)$ and $A_r[k] \leq T_r - k$. Using (2), $T_r - t_r \leq T_r - k$, and $t_r \geq k$, so Basic-P also grants the resource. □

Theorem 3. *The set of reachable states of a system using* Basic-P *is precisely the set of φ_a-states.*

Proof. The proof is analogous to the characterization of the reachable states of Live-P, except that the process P removed in the inductive step is chosen to be a minimal active process in some preference order $>$. This guarantees that P has no children (by the topological property of $>$), and that there is no active process in the same site with lower annotation (by the annotation respecting property of $>$). Consequently, Lemma 5 applies, and the resulting state is also reachable by Basic-P. □

Theorem 3 can also be restated in terms of allocation sequences. For every admissible allocation string that arrives at a φ_a-state there is an admissible allocation string that arrives at the same state and (1) contains no deallocations, and (2) all the nodes occur according to some preference order. It follows from Theorem 3 that $\mathcal{S}(\text{Basic-P}) = \mathcal{S}(\varphi_a)$, and hence, as depicted in Fig 7(b):

$$\mathcal{S}(\text{Basic-P}) = \mathcal{S}(\text{Efficient-P}) = \ldots = \mathcal{S}(k\text{-Efficient-P}) = \ldots = \mathcal{S}(\text{Live-P}).$$

6 Applications and Conclusions

We have generalized our earlier distributed deadlock avoidance algorithms by introducing a family of protocols defined by strengthenings of the global invariant φ. The most liberal protocol, Live-P, also ensures liveness, at the cost of maintaining more complicated data-structures (which require a non-constant number of operations per allocation request). The simplest protocol, Basic-P, can be implemented with one operation per request, but allows less concurrency.

We have shown that all the reachable state spaces of the protocols are the same. This result allows a system designer more freedom in the implementation of a deadlock avoidance protocol, because it follows that every local protocol P that satisfies the following conditions for every request is guaranteed to avoid deadlock:

(1) if Basic-P is enabled then P is enabled, and
(2) if P is enabled then Live-P is enabled

This holds because all P-reachable states satisfy φ, and from those states Basic-P guarantees deadlock freedom. Informally, (2) guarantees that the system stays in a safe region, while (1) ensures that enough progress is made. This result implies, for example, that the combination of different protocols at different sites is safe. If a site has a constraint in memory or CPU time, then the simpler Basic-P is preferable, while Live-P is a better choice if a site needs to maximize concurrency.

This result also facilitates the analysis of alternative protocols. Proving a protocol correct (deadlock freedom) can be a hard task if the protocol must deal with scheduling, external environment conditions, etc. With the results presented in this paper, to show that an allocation manager has no reachable deadlocks, it is enough to map its reachable state space to an abstract system for which all states guarantee φ, and all allocation decisions are at least as liberal as in Basic-P. This technique is used in [5] to create an efficient distributed priority inheritance mechanism where priorities are encoded as annotations, and priority inheritance is performed by an annotation decrease. Although this "annotation decrease" transition is not allowed by the protocols presented here, since the resulting state is still a φ-state, it is also reachable by Basic-P (maybe using a different sequence). Therefore, deadlocks are avoided.

Topics for further research include (1) the question whether Live-P is optimal, that is, does there exist a completely local protocol P that guarantees deadlock

avoidance such that LIVE-P $\sqsubset P$, and (2) the question whether k-EFFICIENT-P is optimal with $O(k \log T_r)$ storage space.

References

1. Andrew D. Birrell. An introduction to programming with threads. Research Report 35, Digital Equipment Corporation Systems Research Center, 1989.
2. Edsger W. Dijkstra. Cooperating sequential processes. Technical Report EWD-123, Technological University, Eindhoven, the Netherlands, 1965.
3. Arie N. Habermann. Prevention of system deadlocks. *Communications of the ACM*, 12:373–377, 1969.
4. James W. Havender. Avoiding deadlock in multi-tasking systems. *IBM Systems Journal*, 2:74–84, 1968.
5. César Sánchez, Henny B. Sipma, Christopher D. Gill, and Zohar Manna. Distributed priority inheritance for real-time and embedded systems. In Alex Shvartsman, editor, *Proceedings of the 10th International Conference On Principles Of Distributed Systems (OPODIS'06)*, volume 4305 of *LNCS*, Bordeaux, France, 2006. Springer-Verlag.
6. César Sánchez, Henny B. Sipma, Zohar Manna, and Christopher Gill. Efficient distributed deadlock avoidance with liveness guarantees. In Sang Lyul Min and Wang Yi, editors, *Proceedings of the* 6^{th} *ACM & IEEE International Conference on Embedded Software (EMSOFT'06)*, pages 12–20, Seoul, South Korea, October 2006. ACM & IEEE.
7. César Sánchez, Henny B. Sipma, Zohar Manna, Venkita Subramonian, and Christopher Gill. On efficient distributed deadlock avoidance for distributed real-time and embedded systems. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS'06)*, Rhodas, Greece, 2006. IEEE Computer Society Press.
8. César Sánchez, Henny B. Sipma, Venkita Subramonian, Christopher Gill, and Zohar Manna. Thread allocation protocols for distributed real-time and embedded systems. In Farn Wang, editor, *25th IFIP WG 2.6 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'05)*, volume 3731 of *LNCS*, pages 159–173, Taipei, Taiwan, October 2005. Springer-Verlag.
9. Douglas C. Schmidt. Evaluating Architectures for Multi-threaded CORBA Object Request Brokers. *Communications of the ACM Special Issue on CORBA*, 41(10):54–60, October 1998.
10. Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley & Sons, New York, 2000.
11. Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, Sixth edition, 2003.
12. Mukesh Singhal and Niranjan G. Shivaratri. *Advanced Concepts in Operating Systems: Distributed, Database, and Multiprocessor Operating Systems*. McGraw-Hill, Inc., New York, NY, 1994.
13. William Stallings. *Operating Systems: Internals and Design Principles*. Prentice Hall, Inc., Upper Saddle River, NJ, Third edition, 1998.
14. Venkita Subramonian, Guoliang Xing, Christopher D. Gill, Chenyang Lu, and Ron Cytron. Middleware specialization for memory-constrained networked embedded systems. In *Proc. of 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*. IEEE Computer Society Press, May 2004.