<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Yamauchi, Yukiko; Tixeuil, Sebastien; Yamashita, Masafumi</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 (2012), 1799: 103-110</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2012-06</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/172994">http://hdl.handle.net/2433/172994</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
<tr>
<td>Textversion</td>
<td>Kyoto University</td>
</tr>
</tbody>
</table>
Probabilistic Stabilization under Probabilistic Schedulers

Yukiko Yamauchi* Sébastien Tixeuil† Masafumi Yamashita‡

概要
Probabilistically stabilizing systems, which are considered to be a probabilistic version of self-stabilizing systems, guarantee that any execution eventually reaches a legitimate execution with probability 1. Unlike self-stabilizing systems, probabilistically stabilizing systems are easy to design, and indeed any weak stabilizing system can be automatically transformed into a probabilistically stabilizing system either by randomizing the algorithm or by introducing a probabilistic scheduler, provided that the number of configurations is finite [Devisines et al., 2008]. In this paper, we discuss how to design a probability distribution $D$ for a given weak stabilizing algorithm to obtain a good probabilistically stabilizing system under an adversarial probabilistic scheduler $M$. Our goodness measure is the convergence time; the expected number of steps $\tau_{D,M}$ necessary to reach a legitimate execution from the worst initial configuration. We then show a necessary and sufficient condition for a $D$ to exist such that $\tau_{D,M} < \infty$ for any $M$ in a wide and natural class of probabilistic schedulers.

1 Introduction
A distributed system consists of a set of processes and algorithms at these processes that make processes cooperate to achieve the specification of the system. Each process maintains its own local state by communicating with neighboring processes, and the specification (task) of the distributed system is achieved by the cooperation among processes. One of the difficulties in designing distributed algorithms is the adversarial behavior of the environment coming from its distributed nature, such as the asynchrony among processes, the communication delay, and the unreliability of communication links. The scheduler abstraction deals with one aspect of this difficulty. A scheduler selects a set of processes that execute their own local algorithm at a given time step. In a deterministic environment, schedulers are considered to be adversarial and algorithms are designed to guarantee its correctness and its performance even in the worst case scenario.

A distributed system is self-stabilizing if, starting from any initial configuration, any execution eventually reaches a configuration after which the system remains in configurations that satisfy system specification [3]. This configuration is called legitimate configuration. Self-stabilizing systems have been attracted attention in the area of fault tolerance because they guarantee convergence to a legitimate configuration irrespective of the initial configuration and regains their specification automatically. The convergence time of a self-stabilizing system is measured by the maximum (the worst case) time necessary for the system to reach a legitimate configuration. There are weaker notions of self-stabilization to avoid the difficulties in designing and proving self-stabilizing algorithms. Prob-
Abilistic stabilization [9] weakens the convergence property. A distributed system is probabilistically stabilizing if, starting from any initial configuration, any execution eventually reaches a legitimate configuration with probability 1. Weak stabilization [7] does not consider the variety of executions caused by the schedulers. A distributed system is weak stabilizing if any configuration has at least one execution that reaches a legitimate configuration. Pseudo stabilization [1] guarantees that every execution has a suffix that satisfies the specification while it does not promise the convergence to a legitimate configuration.

Devismes, Tixeuil, and Yamashita [5] showed that we can translate a weak-stabilizing system under a deterministic fair scheduler to a probabilistically stabilizing system under the uniform probabilistic schedulers. A uniform probabilistic scheduler selects each enabled process with probability 1/2. The paper also showed that uniform randomization of a deterministic algorithm translates a weak stabilizing system under a deterministic fair scheduler to a probabilistically stabilizing system under a synchronous scheduler that selects all processes at each time step. When scheduled, each process flips a coin to choose whether it executes its deterministic algorithm or not. It is not difficult to see that these translations guarantees probabilistic stabilization for non-uniform probabilistic schedulers, non-uniform randomization and the combination of them. This results in the possibility that we can design a “good” probabilistic behavior of a weak stabilizing system, and the existence of an adversarial (the worst) probabilistic scheduler for each randomized stabilizing system.

Our contribution. Motivated by the previous results [5], we investigate how to design “good” probabilistic behavior of a weak stabilizing algorithm under probabilistic schedulers. The randomization of a weak stabilizing algorithm is modeled by probability distribution over the transitions. We consider probabilistic schedulers defined by finite state Markov chains. Our criteria for “goodness” is the expected convergence time, i.e., the expected number of steps from the worst initial configuration to a legitimate configuration. Let \( \tau_{D,M} \) be the expected convergence time of a probabilistically stabilizing system with probability distribution \( D \) of the algorithm and probabilistic scheduler \( M \). We show a necessary and sufficient conditions for a finite system to have \( \tau_{D,M} < \infty \). A system is finite if the set of all configurations is finite. Our result shows that the transition diagram of a system should have regularity property which is newly introduced in this paper.

Related works. Randomized self-stabilizing algorithms are often used for symmetry breaking that is unsolvable deterministically, for example, vertex coloring [8], and token circulation [9, 11]. It is also used to reduce space complexity [10].

Previous papers provide formal models that combines probabilistically stabilizing systems and stochastic processes. Devismes, Tixeuil, and Yamashita [5] used a Markov chain to represent probabilistic behavior of schedulers and randomized algorithms. Any execution of probabilistic system corresponds to a random walk on the transition diagram of the system. There are other techniques to measure the expected convergence time of a probabilistically stabilizing algorithm based on the hitting time of a Markov chain [4], and the coupling technique of Markov chains [6].

Beauquier, Johnen, and Messika [2] used a Markov decision process to represent the behavior of systems under probabilistic schedulers. The probabilistic schedulers are defined by probability distribution that depends on the latest (finite length of) execution and the current configuration.
Different from [2], we assume that schedulers cannot see the execution once it starts.

2 Preliminary

A distributed system is defined by a pair \((N, A)\) of communication graph \(N = (P, L)\), where \(P\) \((|P| = n)\) is the set of processes and \(L\) \((|L| = m)\) is the set of communication links, and an algorithm \(A = \{A_p : p \in P\}\), where \(A_p\) is a (local) algorithm for process \(p\). Process \(p \in P\) is a state machine that maintains local variables specified in \(A_p\). A directed edge \((p, q) \in L\) means that process \(q\) can read the local variables of \(p\). We call \(q\) a predecessor of \(p\). Note that each process \(p\) can read and write to its local variables.

A state of process \(p \in P\) is an assignment of a value to each local variable drawn from its specified domain. Let \(S_p\) be the set of states of \(p\). The set of configurations is the Cartesian product \(\Gamma = \Pi_{p \in P} S_p\). We say that a distributed system is finite if \(\Gamma\) is finite. Since we assume \(P\) is finite, the set of local states at each process is finite, or equivalently, the domain of each variable is finite if and only if \(\Gamma\) is finite.

A deterministic algorithm \(A_p\) is described by a sequence of guarded commands \((\text{guard}) \rightarrow (\text{command})\). In a configuration \(\gamma \in \Gamma\), \(p\) is enabled when at least one of the guards is satisfied, and the corresponding command is executed if its scheduler, which we will define later, activates \(p\). When more than one guard is satisfied in \(\gamma\), the command corresponding to the first enabled guard is executed when the process is activated.

A randomized algorithm \(A_p\) is also described by a sequence of guarded commands but the command executed when it is activated is determined probabilistically. When \(Z\) is the set of guards satisfied at \(p\) in \(\gamma\) and a special symbol \(\perp\), then \(A_p\) is associated with a probability distribution \(D_p^Z\); when \(p\) is activated by the scheduler, \(z \in Z\) is chosen for execution with probability \(D_p^Z(z)\), where \(D_p^Z(\perp)\) is the probability that no guarded command is executed even if \(p\) is enabled in \(\gamma\). Note that \(D_p^Z\) may depend on local information available for \(p\), i.e., the current states of \(p\) and its predecessors \(N_p\). For simplicity, we omit \(Z\) from \(D_p^Z\) whenever it is obvious from the context. Let \(D = \{D_p : p \in P\}\) and we simply call \(D\) a probability distribution (for algorithm \(A\)).

A randomized algorithm is a pair \((A, D)\) where \(A\) is a deterministic algorithm and \(D\) is a probability distribution for \(A\). Probability distribution \(D\) is said to be fair if \(D_p^Z(z) > 0\) for any \(z \in Z\). It is said to be potentially stable if \(D_p^Z(\perp) > 0\) for any \(p\) and \(Z\). We say \(D\) is pure if \(D_p^Z(g) > 0\) implies that \(g\) is the first (in the order of \(A\)) guarded command in \(Z\) or \(g = \perp\) and \(D_p^Z(\perp) < 1\). We denote by \(D\) a pure probability distribution that assigns probability \(\delta\) \((0 < \delta \leq 1)\) to any \(D_p^Z\).

Schedulers. A deterministic scheduler \(\sigma\) is an abstraction of the environment and specifies which process the environment allows to execute at a given time. Hence, \(\sigma\) is a set of infinite sequence of a subset of \(P\).

We denote by \(\sigma_F\) the (strongly) fair scheduler, which is the set of all (strongly) fair sequences, i.e., every process appears infinitely many times in every sequence in \(\sigma_F\). A scheduler is said to be proper if it never selects the empty set to activate.

An execution \(E = \gamma_0, \gamma_1, \ldots\) of a distributed system under scheduler \(\sigma\) starting from an initial configuration \(\gamma_0\) is defined as follows. First, a scheduler non-deterministically selects a sequence \(Z\) from \(\sigma\). For any \(t \geq 0\), let \(X \subseteq P\) be the set of enabled processes in \(\gamma_t\) and \(Z_t \subseteq P\) be the \(t\)-th element of \(Z\), respectively. Then the processes in \(X \cap Z_t\) are activated. If the algorithm is deterministic, then the command that is executed is determined indepen-
dently at each of the processes and their execution yields the next configuration $\gamma_{t+1}$.

If the algorithm is randomized, the command that is executed is selected at random with $\mathcal{D}$, and their executions results in a system transition from $\gamma_{t}$ to the next configuration $\gamma_{t+1}$. Since we cannot control the environment, we consider a scheduler as an adversary and conduct a worst case analysis, assuming that the adversary does not know the results of probabilistic choices at processes a-priori.

A probabilistic scheduler is modeled by a set of Markov chains. A Markov chain $\mathcal{M}$ is a discrete stochastic process $\{X_{t} : t = 0, 1, \ldots\}$ defined by a state space $\Omega = \{1, 2, \ldots \}$ and a transition matrix $P = (P_{i,j})$. In our formalization, $\mathcal{M}$ associates each of the transition with a label $X \subseteq P$ in such a way that no two transitions from a state do not share the same label. We assume that when an execution is going to start, a Markov chain $\mathcal{M}$ and a state $i$ (as the initial state) are non-deterministically selected to specify the behavior of the scheduler. After that, the scheduler cannot see the configuration of the system and the schedule is the sequence of labels attached to transitions that a Markov chain on $\mathcal{M}$ traces.

A probabilistic scheduler is finite if its state space is finite, and is fair if for any subset $X \subseteq P$ (including the empty set) and any state $i$, the probability that $X$ is chosen to be activated (the Markov chain chooses $t_{i}(X)$) at $i$ is positive. In the following, we consider probabilistic schedulers modeled by finite Markov chains. We denote the fair and finite probabilistic scheduler by $\rho_{F}$, i.e., the set of fair and finite Markov chains. We denote the central probabilistic scheduler by $\rho_{c}$, the set of proper Markov chains in $\rho_{F}$ that give positive probability to transitions labeled with $\{p\}$ ($p \in P$) (including the empty set). Another important class is the oblivious (memory-less) scheduler, denoted by $\rho_{O}$, that is the set of Markov chains with a single state. We denote the oblivious central scheduler by $\rho_{OC}$.

Like a deterministic scheduler, we also regard a probabilistic scheduler as an adversary and conduct a worst case analysis, i.e., for a given scheduler $\rho$, we assume an arbitrary Markov chain $\mathcal{M} \in \rho$ is chosen by the scheduler. When the algorithm is randomized, we assume that the adversary does not know the result of the probabilistic choices a-priori.

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{probsched.png}
\caption{Probabilistic finite state schedulers}
\end{figure}

The executions of a distributed system are represented by a transition diagram. For a distributed system executing a deterministic algorithm $\mathcal{A}$ on a communication network $N = (P, L)$, let $\mathcal{S} = (\Gamma, T)$ be a transition diagram where $\Gamma$ is the set of configurations and $T$ is the set of transitions defined by $\mathcal{A}$. Each directed edge $(\gamma; \gamma') \in T$ is labeled by $X \subseteq P$ that means the execution of $\mathcal{A}$ at processes in $X$ in $\gamma$ translates the configuration to $\gamma'$.

An execution of $\mathcal{A}$ under a given scheduler $\sigma$ corresponds to a (in)finite path in $\mathcal{S}$ that satisfies the condition of $\sigma$. Hence, a scheduler removes some of the edges from $\mathcal{S}$. We denote this transition diagram by $\mathcal{S}_{\sigma}$. For example, $\mathcal{S}$ under $\rho_{e}$ is a subgraph of $\mathcal{S}$ where we have transitions labeled with a singleton of $P$.

In a randomized algorithm of $\mathcal{A}$, an enabled guarded command is chosen at random from the set of enabled guarded commands with $\mathcal{D}$, and is executed when the process is activated. Hence, the transition diagram of a distributed system executing $(\mathcal{A}, \mathcal{D})$ on $N$ contains transitions not in $\mathcal{S}$ in
general. (We note that for any pure $D$, the transition diagram does not contain such transitions. ) We denote by $S_D$ this transition diagram with probability distribution $D$. We use $S$ ($S_D$) to refer the corresponding distributed system.

**Self-stabilization.** A specification (task) of an algorithm is a predicate defined over executions. Let $S$ be a distributed system executing algorithm $A$ on a communication graph $N = (P, L)$, and $SP$ be the specification of $A$. We say that $S$ under scheduler $\sigma$ is self-stabilizing if any execution under $\sigma$ contains a legitimate configuration. A configuration of $S$ under $\sigma$ is legitimate if any execution starting from the configuration satisfies $SP$. Here, we denote the set of legitimate configurations by $\Gamma_L$.

We say that $S$ under $\sigma$ for $SP$ is weak stabilizing if any configuration has at least one execution that reaches a legitimate configuration.

We say that $S$ under $\rho$ for $SP$ is probabilistically stabilizing if any execution under $\rho$ reaches a legitimate configuration with probability 1. When we consider a randomized algorithm (i.e., $S_D$), and/or a probabilistic scheduler, the probabilistic stabilization is defined in the same way.

The performance of a stabilizing system is measured by convergence time. When both $A$ and $\sigma$ are deterministic, the convergence time is the maximum (i.e., the worst-case) length of an execution from any configuration to a legitimate configuration.

For a randomized algorithm (a probabilistic scheduler, respectively), we take the expected value of the length of an execution from any configuration to a legitimate configuration. The probability of an execution is the probability that the randomized algorithm (or a probabilistic scheduler) generates the execution. Let $\tau_{D, M}(\gamma)$ be the expected convergence time to a legitimate configuration of $S_D$ when the initial configuration is $\gamma$ and the schedule is generated by a Markov chain $M \in \rho_F$. Define $\tau_{D, M} = \max_{\gamma \in \mathcal{P}} \tau_{D, M}(\gamma)$, and $\tau_D = \max_M \tau_{D, M}$. Then, we want to know $\tau^* = \tau_D$, where $\mathcal{D}^* = \arg\min_D \tau_D$.

Recall that we cannot choose $M$ but can choose $D$ as a part of algorithm design so as the system to have a small $\tau_D$.

**Hitting time of a Markov chain.** In the following, we consider the probabilistic behavior of a distributed system. An execution of a probabilistic (caused by the randomized algorithm and/or the probabilistic scheduler) distributed system is a Markov chain over its transition diagram. In the theory of Markov chains, the time to reach a state from another state is called hitting time. Let $M$ be a Markov chain with state space $\Omega = \{1, 2, \ldots\}$. The hitting time $ht_{i,j}$ is the number of steps that the stochastic process starting from state $i$ takes until it reaches state $j$ for the first time; $ht_{i,j} \equiv \min\{t > 0 : X(t) = j | X(0) = i\}$. The mean hitting time $HT_{i,j}$ is $E[ht_{i,j}]$ and the mean hitting time of $M$, denoted by $HT_M$ is $\max_{i,j \in \Omega} HT_{i,j}$.

### 3 Finite expected convergence time under probabilistic finite schedulers

Let $S$ be a distributed system executing a deterministic algorithm $A$ on a communication network $N$ and suppose that $S$ under $\sigma_F$ is weak stabilizing to $SP$. Then, $S$ under $\rho_F$ is probabilistically stabilizing with any potentially stable and pure distribution $D$ [5]. In this section, we give a necessary and sufficient condition for $S_D$ to have finite expected convergence time under $\rho_F$, i.e., $\tau^* < \infty$. In the following, we consider only potentially stable and pure distribution $D$. 
Let us start with the following weak stabilizing system as an example. Let $N = (P, L)$ where $P = \{p, q\}$ and $L = \{(p, q), (q, p)\}$. Consider a distributed system $S_1$ such that $p$ and $q$ maintain their own local variables $v_p$ and $v_q$, and the transition diagram is the one shown in Figure 2. Because each configuration has a path to the legitimate configuration $(1, 1)$, $S$ is probabilistically stabilizing under $\rho_{OC}$.

Now, we check the expected convergence time of $S_1$ under $\rho_{OC}$. Consider $M \in \rho_0$ that outputs $\{p, q\}$ with probability $\epsilon$, and $\{p\}, \{q\}$ with probability $(1 - \epsilon)/2$. By taking $\epsilon \to 0$, the expected convergence time of $S_1$ becomes arbitrarily large.

![Figure 2: Distributed system $S_1$ (A tuple on each configuration represents $(v_p, v_q)$.)](image)

The contracted transition diagram of $S$ is a digraph $G = (V, E)$ obtained from $S$ by contracting all legitimate configurations to one vertex $\gamma_L$ and removing all vertices reachable from $\gamma_L$.

Let $(V_1, V_2)$ be any cut of $G$ such that $\gamma_L \in V_2$. We denote the directed edges that cross from $V_1$ to $V_2$ by $E(V_1, V_2) = \{(v, v') \in E | v \in V_1, v' \in V_2\}$, and the union of the labels on the edges in $E(V_1, V_2)$ by $P(V_1, V_2)$.

**Lemma 1** For any distributed system $S$, if there is a cut $(V_1, V_2)$ such that $P(V_1, V_2) \neq \{(p) : p \in P\}$ in $G$, then $\tau^* = \infty$.

**Proof.** Suppose that there is a cut $(V_1, V_2)$ such that $P(V_1, V_2) \neq \{(p) : p \in P\}$ and let $\{p\} \notin P(V_1, V_2)$. Consider a Markov chain $M$ in $\rho_{OC}$ that assigns probability $(1 - \epsilon)$ to $\{p\}$ for arbitrary small $\epsilon$. Consider an execution starting from a configuration in $V_1$. Then, the expected number of steps necessary to cross this cut is $\epsilon^{-1}$. Hence, the maximum convergence time is at least $\epsilon^{-1}$. For any $D, M$ makes $\tau^*$ arbitrarily large.

We have the same argument when there are multiple processes whose singleton is not in $P(V_1, V_2)$.

In order for $\tau^*$ to have a finite value, for any cut $(V_1, V_2)$, $P(V_1, V_2) = \{(p) : p \in P\}$ must hold, which implies that in any configuration $\gamma$, all processes in $P$ are enabled unless $\gamma \in \Gamma_L$.

Let $E_p$ be the set of edges in $G$ labeled with $\{p\}$. From Lemma 1, for any $p \in P$, the subgraph $S_p = (V, E_p)$ is 1-regular in the sense that for any state except $\gamma_L$, the out degree is exactly one. It is known that $S_p$ forms a rooted in-tree rooted at $\gamma_L$ if and only if it is weakly connected. Otherwise, $S_p$ consists of multiple connected components, and we have $\tau^* = \infty$ by taking a cut that separates these connected components.

**Lemma 2** If $S_p$ is not a single rooted in-tree for some $p \in P$, then $\tau^* = \infty$.

We say that $S$ satisfies regularity condition if $S_p$ is a single rooted in-tree rooted at $\gamma_L$ for any $p \in P$.

In the following, we show that regularity condition is a necessary and sufficient condition for $\tau^*$ to be finite under $\rho_{OC}$. Let $G$ be a Markov chain obtained from $G$ by assigning to a transition labeled with $X \subseteq P$ the probability of the corresponding transition (i.e., the transition labeled with $X$) in $M \in \rho_{OC}$.

**Theorem 3** $S$ satisfies the regularity condition if and only if $\tau^* < \infty$ under $\rho_{OC}$.
Proof. If part is proved by Lemma 1. We will show the sketch of the proof for the only-if part by showing that \( \tau_{D_{1}} = \max_{M \in \rho_{OC}} \tau_{D_{1},M} \geq \tau^{*} \) is finite. Let \( p \) be a process that \( M \in \rho_{OC} \) outputs with the maximum probability \( \delta \). Let \( h \) be the height of \( S_{p} \). The event that an execution traverses \( S_{p} \) to reach \( \gamma_{L} \) takes at most \( h \) steps and this event occurs with probability more than \( \delta \). Because the probability that this event does not occur during an execution decreases exponentially, and such \( p \) exists for any \( M \in \rho_{OC} \), we have \( \tau_{D_{1}} = \max_{M \in \rho_{OC}} \tau_{D_{1},M} \geq \tau^{*} < \infty \).

Next, we consider \( \rho_{O} \) that activates any subset \( X \in P \). If \( S \) under \( \rho_{O} \) consists of transitions corresponding to an execution of multiple processes, then when given \( M \in \rho_{OC} \subset \rho_{O} \), the system remains in an initial configuration. Hence, in order to have \( \tau^{*} < \infty \), it is necessary that \( S \) under \( \rho_{OC} \) should contain paths from any configuration to a legitimate configuration such that each of the transitions is labeled with a singleton. On the other hand, \( \rho_{OC} \) may always activate multiple processes. By using \( D_{1}/P_{1} \), we can probabilistically produce a centralized schedule and the regularity condition promises the expected convergence time is finite. Then we have the following theorem. (We omit the detailed proof due to space restriction.)

**Theorem 4** \( S \) satisfies the regularity condition if and only if \( \tau^{*} < \infty \) under \( \rho_{O} \).

When we remove the assumption of obliviousness, there is a deterministic distributed system where we have \( \tau^{*} = \infty \). Consider a distributed system \( S \) shown in Figure 3 executing algorithm \( A \) on \( N = (P,L) \) where \( P = \{p,q\} \) and \( L = \{(p,q),(q,p)\} \). Process \( p \) maintains a variable \( v_{p} \) (\( v_{q} \) at \( q \), respectively) that takes an integer in \( \{0,1,2,3\} \). The legitimate configurations are the configurations where \( v_{p} = v_{q} = 1,2,3 \). The transitions of \( S \) is represented by a state machine shown in Figure 4 where \( s_{4} \) corresponds to the legitimate configurations.

Let \( M \in \rho_{OC} \) be a Markov chain with two states 1 and 2 such that the transition \( P_{1,2} = P_{2,1} = 1 - \epsilon \), transitions \((1,2),(2,2)\) are labeled with \( \{p\} \), and transitions \((2,1),(1,1)\) are labeled with \( \{q\} \). Starting from an initial configuration where \( v_{p} = v_{q} = 0 \), the expected convergence time becomes arbitrarily large by taking arbitrarily small \( \epsilon \) which makes \( M \) outputs \( \{q\}\{p\}\{q\}\{p\}\{q\}\ldots \) with arbitrarily high probability. To overcome this effect, we use \( D_{1}/2 \) to ignore some activations and probabilistically get out of such loops. This strategy works for any finite schedulers to probabilistically produce executions that follows \( S_{p} \) of a process \( p \in P \). Then we have the following theorem. (We omit the detailed proof due to space restriction.)

**Figure 3:** Transition diagram \( S \) (Each tuple represents \((v_{p},v_{q})\))

**Figure 4:** A state machine corresponding to \( S \)
Theorem 5 $S$ satisfies the regularity condition if and only if $\tau^* < \infty$ under $\rho_C$.

Any scheduler $\mathcal{M}$ in $\rho_F$ is represented by finite state fair Markov chains. Hence, $\mathcal{M}$ has the property of concurrent activation and non-obliviousness. We showed that $D_{1/|P|}$ resolves the concurrency (Theorem 4), and $D_{1/2}$ avoids the memory of the scheduler. Hence, we obtain the following theorem from Theorem 4 and Theorem 5.

Theorem 6 $S$ satisfies the regularity condition if and only if $\tau^* < \infty$ under $\rho_F$.

4 Conclusion

In this paper, we investigate the power of randomization of an algorithm against the probabilistic behavior of schedulers. We showed necessary and sufficient conditions for finite probabilistically stabilizing systems to have finite expected convergence time. Except for oblivious central schedulers, randomization is necessary to guarantee finite expected convergence time. Our future work is to obtain an optimal randomization with the minimum expected convergence time.

参考文献


