数理解析研究所講究録
第 1799 巻 2012 年 103-110

2011 年度冬の LA シンポジウム [21]

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 [Devismes 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 satisfies 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 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-

*Kyushu University, Japan
†Paris 6, France
‡Kyushu University, Japan

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 $\tau_{\mathcal{D},\mathcal{M}}$ be the expected convergence time of a probabilistically stabilizing system with probability distribution $\mathcal{D}$ of the algorithm and probabilistic scheduler $\mathcal{M}$. We show a necessary and sufficient conditions for a finite system to have $\tau_{\mathcal{D},\mathcal{M}} < \infty$. 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 $(N, \mathcal{A})$ 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 $\mathcal{A} = \{A_p : p \in P\}$, 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 $\Gamma = \prod_{p \in P} S_p$. We say that a distributed system is finite if Γ 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 Γ is finite.

A *deterministic algorithm* A_p is described by a sequence of guarded commands $\langle \text{guard} \rangle \rightarrow \langle \text{command} \rangle$. 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 γ, 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 γ 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 γ. 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 $\mathcal{D} = \{D_p : p \in P\}$ and we simply call $\mathcal{D}$ a *probability distribution* (for algorithm $\mathcal{A}$).

A randomized algorithm is a pair $\langle \mathcal{A}, \mathcal{D} \rangle$ where $\mathcal{A}$ is a deterministic algorithm and $\mathcal{D}$ is a probability distribution for $\mathcal{A}$. Probability distribution $\mathcal{D}$ 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 $\mathcal{D}$ is *pure* if $D_p^Z(g) > 0$ implies that g is the first (in the order of $\mathcal{A}$) guarded command in Z or $g = \perp$ and $D_p^Z(\perp) < 1$. We denote by $\mathcal{D}_\delta$ a pure probability distribution that assigns probability δ ($0 < \delta \leq 1$) to any D_p^z.

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

We denote by σ_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 σ_F. A scheduler is said to be *proper* if it never selects the empty set to activate.

An *execution* $\mathcal{E} = \gamma_0, \gamma_1, \ldots$ of a distributed system under scheduler σ starting from an initial configuration γ_0 is defined as follows. First, a scheduler non-deterministically selects a sequence Z from σ. For any $t \geq 0$, let $X \subseteq P$ be the set of enabled processes in γ_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 independently at each of the processes and their execution yields the next configuration γ_{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 γ_t to the next configuration γ_{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 ρ_F, *i.e.*, the set of fair and finite Markov chains. We denote the central probabilistic scheduler by ρ_C, the set of proper Markov chains in ρ_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 ρ_O, that is the set of Markov chains with a single state. We denote the *oblivious central* scheduler by ρ_{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 ρ, 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.

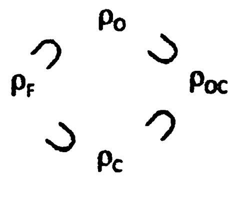

図 1: Probabilistic finite state schedulers

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 Γ 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 γ translates the configuration to γ'.

An execution of $\mathcal{A}$ under a given scheduler σ corresponds to a (in)finite path in $\mathcal{S}$ that satisfies the condition of σ. Hence, a scheduler removes some of the edges from $\mathcal{S}$. We denote this transition diagram by $\mathcal{S}$ *under* σ. For example, $\mathcal{S}$ under ρ_c 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 $\langle \mathcal{A}, \mathcal{D} \rangle$ on N contains transitions not in $\mathcal{S}$ in general. (We note that for any pure $\mathcal{D}$, the transition diagram does not contain such transitions.) We denote by $\mathcal{S}_{\mathcal{D}}$ this transition diagram with probability distribution $\mathcal{D}$. We use $\mathcal{S}$ ($\mathcal{S}_{\mathcal{D}}$) to refer the corresponding distributed system.

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

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

We say that $\mathcal{S}$ under ρ for $\mathcal{SP}$ is *probabilistically stabilizing* if any execution under ρ reaches a legitimate configuration with probability 1. When we consider a randomized algorithm (*i.e.*, $\mathcal{S}_{\mathcal{D}}$), 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 $\mathcal{A}$ and σ 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 $\tau_{\mathcal{D},\mathcal{M}}(\gamma)$ be the expected convergence time to a legitimate configuration of $\mathcal{S}_{\mathcal{D}}$ when the initial configuration is γ and the schedule is generated by a Markov chain $\mathcal{M} \in \rho_F$. Define $\tau_{\mathcal{D},\mathcal{M}} = \max_{\gamma\in\Gamma} \tau_{\mathcal{D},\mathcal{M}}(\gamma)$, and $\tau_{\mathcal{D}} = \max_{\mathcal{M}} \tau_{\mathcal{D},\mathcal{M}}$. Then, we want to know $\tau^* = \tau_{\mathcal{D}^*}$ where $\mathcal{D}^* = \mathrm{argmin}_{\mathcal{D}} \tau_{\mathcal{D}}$.

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

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 $\mathcal{M}$ 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 $\mathrm{E}[ht_{i,j}]$ and the mean hitting time of $\mathcal{M}$, denoted by $HT_{\mathcal{M}}$ is $\max_{i,j\in\Omega} HT_{i,j}$.

3 Finite expected convergence time under probabilistic finite schedulers

Let $\mathcal{S}$ be a distributed system executing a deterministic algorithm $\mathcal{A}$ on a communication network N and suppose that $\mathcal{S}$ under σ_F is weak stabilizing to $\mathcal{SP}$. Then, $\mathcal{S}$ under ρ_F is probabilistically stabilizing with any potentially stable and pure distribution $\mathcal{D}$ [5]. In this section, we give a necessary and sufficient condition for $\mathcal{S}_{\mathcal{D}}$ to have finite expected convergence time under ρ_F, *i.e.*, $\tau^* < \infty$. In the following, we consider only potentially stable and pure distribution $\mathcal{D}$.

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 $\mathcal{S}_1$ 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 $(1, 1)$, $\mathcal{S}$ is probabilistically stabilizing under ρ_{OC}.

Now, we check the expected convergence time of $\mathcal{S}_1$ under ρ_{OC}. Consider $\mathcal{M} \in \rho_O$ that outputs $\{p, q\}$ with probability ϵ, and $\{p\}, \{q\}$ with probability $(1-\epsilon)/2$. By taking $\epsilon \to 0$, the expected convergence time of $\mathcal{S}_1$ becomes arbitrarily large.

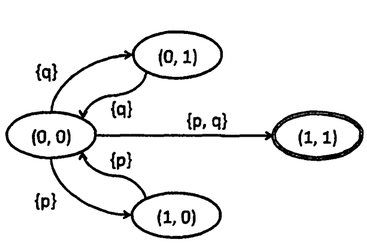

図 2: Distributed system $\mathcal{S}_1$ (A tuple on each configuration represents (v_p, v_q).)

The *contracted transition diagram* of $\mathcal{S}$ is a digraph $G = (V, E)$ obtained from $\mathcal{S}$ by contracting all legitimate configurations to one vertex γ_L and removing all vertices reachable from γ_L[1].

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

[1]Because we are interested in convergence of $\mathcal{S}$ under ρ, we do not consider the executions from γ_L that always satisfies $\mathcal{SP}$.

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 $\mathcal{M}$ in ρ_{OC} that assigns probability $(1-\epsilon)$ to $\{p\}$ for arbitrary small ϵ. Consider an execution starting from a configuration in V_1. Then, the expected number of steps necessary to cross this cut is ϵ^{-1}. Hence, the maximum convergence time is at least ϵ^{-1}. For any $\mathcal{D}$, $\mathcal{M}$ makes τ^* 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 τ^* 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 γ, 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 $\mathcal{S}_p = (V, E_p)$ is 1-regular in the sense that for any state except γ_L, the out degree is exactly one. It is known that $\mathcal{S}_p$ forms a rooted in-tree rooted at γ_L if and only if it is weakly connected. Otherwise, $\mathcal{S}_p$ consists of multiple connected components, and we have $\tau^* = \infty$ by taking a cut that separates these connected components.

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

We say that $\mathcal{S}$ satisfies *regularity condition* if $\mathcal{S}_p$ is a single rooted in-tree rooted at γ_L for any $p \in P$.

In the following, we show that regularity condition is a necessary and sufficient condition for τ^* to be finite under ρ_{OC}. Let $\mathcal{G}$ 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 $\mathcal{M} \in \rho_{OC}$.

Theorem 3 *$\mathcal{S}$ satisfies the regularity condition if and only if $\tau^* < \infty$ under ρ_{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 $\tau_{\mathcal{D}_1} = \max_{\mathcal{M} \in \rho_{OC}} \tau_{\mathcal{D}_1, \mathcal{M}} \geq \tau^*$ is finite. Let p be a process that $\mathcal{M} \in \rho_{OC}$ outputs with the maxumum probability δ. Let h be the height of S_p. The event that an execution traverses S_p to reach γ_L takes at most h steps and this event occurs with probability more than $delta^h$. Because the probability that this event does not occur during an execution decreases exponentially, and such p exists for any $\mathcal{M} \in \rho_{OC}$, we have $\tau_{\mathcal{D}_1} = \max_{\mathcal{M} \in \rho_{OC}} \tau_{\mathcal{D}_1, \mathcal{M}} \geq \tau^* < \infty$. □

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

Theorem 4 *$\mathcal{S}$ satisfies the regularity condition if and only if $\tau^* < \infty$ under ρ_O.*

When we remove the assumption of obliviousness, there is a deterministic distributed system where we have $\tau^* = \infty$. Consider a distributed system $\mathcal{S}$ shown in Figure 3 executing algorithm $\mathcal{A}$ 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 $\mathcal{S}$ is represented by a state machine shown in Figure 4 where s_4 corresponds to the legitimate configurations.

Let $\mathcal{M} \in \rho_C$ 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 ϵ which makes $\mathcal{M}$ outputs $\{q\}\{p\}\{q\}\{p\}\{q\}\ldots$ with arbitrarily high probability. To overcome this effect, we use $\mathcal{D}_{1/2}$ to ignore some activations and probabilistically get out of such loops. This strategy works for any finite schedulers to proabilistically produce executions that follows S_p of a process $p \in P$. Then we have the following theorem. (We omit the detailed proof for space restriction.)

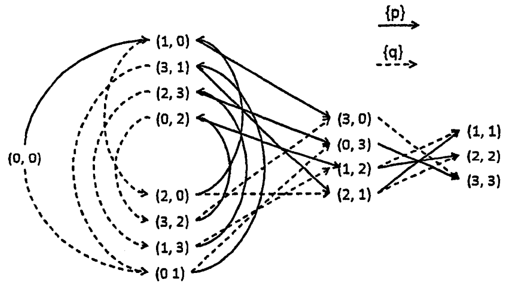

図 3: Transition diagram $\mathcal{S}$ (Each tuple represents (v_p, v_q))

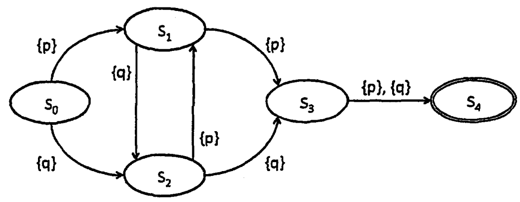

図 4: A state machine corresponding to $\mathcal{S}$

Theorem 5 *$\mathcal{S}$ satisfies the regularity condition if and only if $\tau^* < \infty$ under ρ_C.*

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

Theorem 6 *$\mathcal{S}$ satisfies the regularity condition if and only if $\tau^* < \infty$ under ρ_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

[1] J.E. Burns, M.G. Gouda, and R.E. Miller, Stabilization and pseudo stabilization, Distributed Computing, **7**(1), pp.35–42, 1993.

[2] J. Beauquier, C. Johnen, and S. Messika, All k-bounded policies are equivalent for self-stabilization, In Proc. of SSS, pp.82–94, 2006.

[3] E.W. Dijkstram Self-stabilizing systems in spite of distributed control, Communications of ACM, **17**(11), pp.643–644, 1974.

[4] M. Duflot, L. Fribourg, and C. Picaronny, Randomized finite-state algorithms as Markov chains, In Proc. of DISC, pp.240–254, 2001.

[5] S. Devismes, S. Tixeuil, and M. Yamashita, Weak vs. Self vs. Probabilistic stabilization, In Proc. of ICDCS, pp. 681–688, 2008.

[6] L. Fribourg, S. Messika, and C. Picaronny, Coupling and self-stabilization, Distributed Computing, **18**, 3, pp.221–232, 2005.

[7] M.G. Gouda, The theory of weak stabilization, In Proc. of WSS, pp.114–123, 2001.

[8] M. Gradinariu and S. Tixeuil, Conflict managers for self-stabilization without fairness assumption, In Proc. of ICDCS, pp.46–46, 2007.

[9] T. Herman, Probabilistic stabilization, Information Processing Letters, **35**(2), pp.63–67, 1990.

[10] T. Herman, Self-stabilization: randomness to reduce space, Distributed Computing, **6**, pp.95–98, 1992.

[11] A. Israeli, and M. Jafron, Token management schemes and random walks yields self stabilizing mutual exclusion, In Proc. of PODC, pp.119-131, 1990.