Journal of Software Engineering and Applications, **2011, 1, 9-17**
doi:10.4236/jsea.2011.41002 Published Online January 2011 (http://www.scirp.org/journal/jsea)

An Extension to Pi-Calculus for Performance Evaluation

Shahram Rahimi, Elham S. Khorasani, Yung-Chuan Lee, Bidyut Gupta

Department of Computer Science, Southern Illinois University, Illinois, USA.
Email: {rahimi, elhams, yclee, bidyut}@siu.edu

Received October 12th, 2010; revised November 25th, 2010; accepted November 30th, 2010.

ABSTRACT

Pi-Calculus is a formal method for describing and analyzing the behavior of large distributed and concurrent systems. Pi-calculus offers a conceptual framework for describing and analyzing the concurrent systems whose configuration may change during the computation. With all the advantages that pi-calculus offers, it does not provide any methods for performance evaluation of the systems described by it; nevertheless performance is a crucial factor that needs to be considered in designing of a multi-process system. Currently, the available tools for pi-calculus are high level language tools that provide facilities for describing and analyzing systems but there is no practical tool on hand for pi-calculus based performance evaluation. In this paper, the performance evaluation is incorporated with pi-calculus by adding performance primitives and associating performance parameters with each action that takes place internally in a system. By using such parameters, the designers can benchmark multi-process systems and compare the performance of different architectures against one another.

Keywords: *Pi-calculus, Performance Evaluation, Multi-Agent Systems, System Modeling*

1. Introduction

Pi-calculus, introduced by Robin Milner [1-5], stands out as one of the most powerful modeling tools for modeling and analyzing the behavior of concurrent computational systems such as: multi agent systems, grid computing systems, and so forth. Pi-calculus defines the syntax of processes and actions and provides transition rules for analyzing and validating system functionalities. Mobility in pi-calculus is achieved by sending and receiving channel names through which the recipient can communicate with the sender or even with other processes, provided the channel is a free name. With the interaction between the processes changing, the configuration of the whole system is also changed and mobility is attained. In the higher-order pi-calculus [5], computational entities (such as objects, process, or a code segment) can be transferred as well as the values and channel names. Because of its strong support for mobility, higher order pi-calculus is very flexible and powerful in modeling mobile agent systems. As mentioned before, however, pi-calculus by itself does not provide any method for performance evaluation of systems described by it; nevertheless, performance is an important factor that needs to be considered in designing of a multi-process system. The traditional software development methods delay the performance test until after the implementation is completed. This approach has its obvious drawback as if the system is found not satisfying the desired performance requirements, then most of the implementation has to be redone. So evaluating the performance at the design stage has significant importance and can save much of the developers' time, energy and cost.

The previous works for integrating the process algebra with performance measures have either extended the syntax and semantics of the pi-calculus to account for performance, or derived quantitative performance measures from the system transitions.

In the former approach, a family of stochastic process algebra [6-14] has been introduced that extends the standard process algebra with timing or probability information. In this algebra, every action has an associated duration which is assumed to be a random variable with an exponential distribution. A prefix then is in the form of $(a,r).P$, where a represents an action with a duration that is exponentially distributed with parameter r, known as the action rate. The memory-less property of the exponential distribution conforms to the Markov property thus Markov chains [15-16] are applied to compute the probability of reaching each state. Consequently, the final performance of the system can be calculated given the reward model. One potential problem with providing explicit action rates is that when the action rates are not known in advance, a different symbol is used for each action rate which leads to a heavy computational overhead in performance calculation.

In the latter approach [17-18], labels of the transition systems are enhanced to carry additional information about the inference rule which was applied to the transition. So instead of giving an explicit action rate for each action, the probabilistic distributions are computed from the transition labels by associating a cost function to each transition. Then Markov chain is applied as before to calculate the final performance.

In this paper we follow the latter approach, in that we leave the syntax of the pi-calculus intact and associate performance parameters with each inference rule. However, our approach differs from the previous works in two aspects: first, the cost functions proposed in the previous works are too abstract to be directly applied to real cases. Thus, in this paper a cost function is replaced by a concrete transition rate function that estimates the duration time for each transition rule. Second, our transition rates account for process mobility such as transferring objects or codes from one environment to another, considering the fact that the process mobility (such as mobile agents) is different from simple name exchange in its complexity and additional overhead [19].

The rest of the paper is organized as follows: we start with some background knowledge on pi-calculus syntax and its transition rules. Then we review the continuous time Markov chains, their steady state analysis and reward models. Finally, we introduce the transition rate function and establish our performance evaluation methodology based on it. A conclusion section summaries the paper at the end and gives a prospective for future work in this area.

2. Pi-Calculus

Pi-calculus, an extension of the process algebra CCS, was introduced by Robin Milner in the late 80 s [1]. It provides a formal theory for modeling mobile systems and reasoning about their behaviors. In pi-calculus, two entities are specified, “names” and “processes” (or “agents”). “Name” is defined as a channel or a value that is transferred via a channel. We follow the naming conventions and the syntax of [1] in which *u, v, w, x, y, z* range over names and *A, B, C,* $\cdots$ range over process (agent) identifiers.

The syntax of an agent (or a process) in pi-calculus is defined as follows:

$$P ::= 0 \,\big|\, \textstyle\sum_{i\in I}\lambda_i P_i \,\big|\, \bar{y}x.P \,\big|\, \bar{y}(x).P \,\big|\, y(x).P \,\big|\, \tau.P \,\big|\, P_1 \big| P_2 \,\big| \\ P_1 + P_2 \,\big|\, (x)P \,\big|\, [x=y]P \,\big|\, A(y_1,\cdots,y_n) \,\big|\, !P$$

- 0 is a null process and means that agent *P* does not do anything.
- $\sum_{i\in I}\lambda_i.P_i$: Agent *P* will behave as either one of $\lambda_i.P_i$ where $i \in I$, but not more than one, and thebehaves like P_i. If $I = \Phi$, then *P* behaves like 0.
- Here λ_i denotes any actions that could take place in *P* (such as τ, $\bar{y}(x)$, and so on).
- Here λ_i denotes any actions that could take place in *P* (such as τ, $\bar{y}(x)$, and so on).
- $\bar{y}x.P$: Agent *P* sends the free name *x* out along channel *y* and then behaves like *P*.
- Name *x* is said to be free if it is not bound to agent *P*. In the same way, if *x* is bound to *P* (also say private to *P*), that means *x* can only be used inside by *P*.
- $\bar{y}(x).P$: Agent *P* sends bound name *x* out along channel *y* and then behaves like *P*.
- $y(x).P$: Agent *P* receives name *x* along channel *y* and then behaves like *P*.
- $\tau.P$: Agent *P* performs the silent action τ and then behaves like *P*.
- $P_1|P_2$: Agent *P* has P_1 and P_2 executing in parallel. P_1 and P_2 may behave independently or they may interact with each other. E.g. if $P_1 = \tau_1.P_1$ and $P_2 = \tau_2.P_2$, then P_1 and P_2 will behave independently. But, if $P_1 = \bar{y}(x).P_1$ and $P_2 = y(x).P_2$ then P_1 sends name *x* to P_2 via channel *y*.
- The sum $P_1 + P_2$ means either P_1 or P_2 is executed, but not both.
- $(x)P$: is called restriction and means that agent *P* does not change except for that *x* in *P* becomes private to *P*. That means any outside communication through channel *x* is prohibited.
- A match operator [*x* = *y*] *P* means that if *x* = *y*, then the agent behaves like *P*, otherwise it behaves like 0.
- $A(y_1,\cdots,y_n)$ is an agent identifier where $y_1,\cdots,y_n$ are free names occurring in *P*.
- $!P$ is called replication and can be thought of as an infinite composition $P|P|P|\cdots$.

The semantics of the calculus is defined by a set of transition rules which establish the system evolution. A transition rule is in the form of $P \xrightarrow{\alpha} P'$, which means that process *P* evolves to P' after performing action α. Action α can be one of the following prefixes:

$$\alpha ::= \bar{y}x \,|\, \bar{y}(x) \,|\, y(x) \,|\, \tau$$

Pi-calculus transition rules are listed in **Table 1**. The TAU-ACT, INPUT-ACT, OUTPUT-ACT, SUM and MATCH rules correspond directly to the definition of the silent, input, and output actions and the sum and match operators. The side condition $w \notin fn((z)P)$ in the INPUT-ACT rule ensures that a free occurrence of *w*

Table 1. Pi-calculus transition rules [1].

TAU-ACT:	$\dfrac{-}{\tau.P \xrightarrow{\tau} P}$	
OUTPUT-ACT:	$\dfrac{-}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$	
INPUT-ACT:	$\dfrac{}{x(z).P \xrightarrow{x(w)} P}$	$w \notin fn((z)P)$
SUM:	$\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	
MATCH:	$\dfrac{P \xrightarrow{\alpha} P'}{[x=x]P \xrightarrow{\alpha} P'}$	
IDE:	$\dfrac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P'}{A(\tilde{y}) \xrightarrow{\alpha} P'}$	$A(\tilde{x}) \overset{def}{=} P$
PAR:	$\dfrac{P \xrightarrow{\alpha} P'}{P \vert Q \xrightarrow{\alpha} P' \vert Q}$	$bn(\alpha) \cap fn(Q) = \varphi$
COM:	$\dfrac{P \xrightarrow{\alpha} P'}{P \vert Q \xrightarrow{\alpha} P' \vert Q} \dfrac{P \xrightarrow{\bar{x}y} P' \quad Q \xrightarrow{x(z)} Q'}{P \vert Q \xrightarrow{\tau} P' \vert Q'\{y \vert z\}}$	
RES:	$\dfrac{P \xrightarrow{\alpha} P'}{(y)P \xrightarrow{\alpha} (y)P'}$	$y \notin n(\alpha)$
CLOSE:	$\dfrac{P \xrightarrow{\bar{x}(w)} P' \quad Q \xrightarrow{x(w)} Q'}{P \vert Q \xrightarrow{\tau} (w)(P' \vert Q')}$	
OPEN:	$\dfrac{P \xrightarrow{\bar{x}y} P'}{(y)P \xrightarrow{x(w)} P'\{w \vert y\}}$	$y \neq x$ $w \notin fn((y)P')$

does not become bounded by the substitution{w/z}. The PAR rule states that the parallel composition does not inhibit computation; the side condition $bn(\alpha) \cap fn(Q) = \varphi$ guarantees that a free name in *Q* does not turn into a bound name by the application of PAR. The COM rule shows the synchronous communication between two processes. The RES rule states that the process *P* can resume its computation under restriction. The side condition $y \notin n(\alpha)$ assures that no conflict will occur between the restricted name y and the names in *P*. In the CLOSE rule, *P* sends a bound name *w* to *Q* hence it extends the scope of *w* and makes it available to both *P* and *Q*. the Open rule transforms a free output action $\bar{x}y$ to a bound output $\bar{x}(w)$ and eliminates the restriction (*y*). The side condition of the open rule certifies that the bound name *w* may not occur free in P'.

The higher-order pi-calculus is different from the first-order pi-calculus in its ability to model sending and receiving processes as well as the values and channel names. In higher-order pi-calculus, $\bar{x}(K)$ means sending a name or a process *K* via channel *x*, and $x(U)$ means receiving a name or a process via channel *x*. Because of its support for process mobility, we chose higher order pi-calculus as our target modeling language. The transition rules of the higher order pi-calculus are listed in **Table 2**. These rules are basically the same as the transition rules of the first order pi-calculus with an additional rule for process communication (COM-PROCESS).

Table 2. Higher-order pi-calculus labeled transition system [5].

TAU:	$\dfrac{-}{\tau.P \xrightarrow{\tau} P}$	
SUM:	$\dfrac{P \xrightarrow{\tau} P'}{P+Q \xrightarrow{\tau} P'}$	
PAR:	$\dfrac{P \xrightarrow{\tau} P'}{P \vert Q \xrightarrow{\tau} P' \vert Q}$	
MATCH:	$\dfrac{P \xrightarrow{\tau} P'}{[x=x]P \xrightarrow{\tau} P'}$	
IDE:	$\dfrac{P\{\tilde{K} \vert \tilde{U}\}P'}{A(\tilde{K}) \xrightarrow{\tilde{U}} P'}$, if	$A(\tilde{U}) \overset{def}{=} P$
RES:	$\dfrac{P \xrightarrow{u} P'}{(x)P \xrightarrow{u} (x)P'}$	$x \notin n(u)$
COM-NAME:	$\dfrac{P \xrightarrow{\bar{x}\tilde{y}} P' \quad Q \xrightarrow{x(\tilde{z})} Q'}{P \vert Q \xrightarrow{\tau} P' \vert Q'\{\tilde{y} \vert \tilde{z}\}}$	
	$\tilde{y}$ is not a process list.	
COM-PROCESS:	$\dfrac{P \xrightarrow{(\tilde{y})\bar{x}\langle\tilde{K}\rangle} P' \quad Q \xrightarrow{x\langle\tilde{K}\rangle} Q'}{P \vert Q(\tilde{y}) \xrightarrow{\tau} (P' \vert Q')}$	
	$\tilde{K}$ is a process vector.	
CLOSE-NAME:	$\dfrac{P \xrightarrow{\bar{x}(\tilde{w})} P' \quad Q \xrightarrow{x(\tilde{w})} Q'}{P \vert Q \xrightarrow{\tau} (w)(P' \vert Q)'}$	
	$\tilde{w} \notin fn(Q)$ *w* is not a process	
OPEN:	$\dfrac{P \xrightarrow{(\tilde{y})\bar{z}\langle\tilde{K}\rangle} P'}{(x)P \xrightarrow{(x,\tilde{y})\bar{z}\langle\tilde{K}\rangle} P'}$	$x \neq z, x \in fn(\tilde{K})$ $\tilde{y}$

3. Continuous Time Markov Chains

Markov chains are the most commonly used mathematical tools for performance and reliability evaluation of dynamic systems. A Markov chain models a dynamic system with a set of states and a set of labeled transitions between the states. Each label shows the probability of a transitions (in case of the discrete-time Markov chain), or it shows the rate of a transitions (in case of the continuous Markov chain). In this section, we review some basic definitions and theorems from [16] of the stochastic processes and the continuous time Markov chain.

Definition 3.1 (Stochastic Process): A stochastic process is defined as a family of random variables $X = \{X_t : t \in T\}$ where each random variable X_t is indexed by parameter $t \in T$, which is usually called the time parameter if $T \subseteq R_+ = [0, \infty)$. The set of all possible values of X_t (for each $t \in T$) is known as the state space of the stochastic process. X_t is the state of a process at time *t*. If the index set *T* is a countable set, then *X* is a discrete-time stochastic process, otherwise it is a continuous-time process.

Definition 3.2 (Continuous Time Markov Chain

(CTMC)): A stochastic process $\{X_t : t \in T\}$ constitutes a CTMC if it satisfies the Markov property, namely that, given the present state, the future and the past states are independent. Formally, for an arbitrary $t_i \in \mathrm{R}_0^+$ with $0 = t_0 < t_1 < \cdots < t_n < t_{n+1}, \forall n \in N$ and $\forall s \in S = \mathrm{N}_0$, the following relation holds for the conditional probability mass function (pmf):

$$P\left(X_{t_{n+1}} = s_{n+1} \middle| X_{t_n} = s_n, X_{t_{n\,1}} = s_{n\,1}, \cdots, X_{t_0} = s_0\right) = P\left(X_{t_{n+1}} = s_{n+1} \middle| X_{t_n} = s_n\right)$$

where $P\left(X_{t_{n+1}} = s_{n+1} \middle| X_{t_n} = s_n\right)$ is the transition probability to travel from state S_n to state S_{n+1} during the period of time $[t_n, t_{n+1}]$.

Definition 3.3 (Time-homogenous CTMC): A CTMC is called time-homogenous if the transition probabilities $P\left(X_{t_{n+1}} = s_{n+1} \middle| X_{t_n} = s_n\right)$ depend only on the time difference t_{n+1}-t_n, and not on the actual values of t_{n+1} and t_n.

Definition 3.4 (State Sojourn Time): The sojourn time of a state is the time spent in that state before moving to another state. The sojourn time of time-homogenous CTMC exhibits the memory less property. Since the exponential distribution is the only continuous distribution with this property, the random variables denoting the sojourn times, or holding times, must be exponentially distributed.

Definition 3.5 (Irreducible CTMC): A CTMC is called irreducible if every state S_i is reachable from every other state S_i, that is:

$$\forall S_i, S_j, S_i \neq S_j, \exists t > 0 : p_{i,j}(t) > 0$$

Definition 3.6 (Instantaneous Transition Rates of CTMC): A transition rate $q_{ij}(\mathrm{t})$, measures how quickly a process travels from state *i* to state *j* at time *t*. The transition rates are related to the conditional transition probabilities as follows:

$$q_{i,j}(t) = \begin{cases} \lim\limits_{\Delta t \to 0} \dfrac{p_{i,j}(t, t+\Delta t)}{\Delta t}, & i \neq j \\ \lim\limits_{\Delta t \to 0} \dfrac{p_{i,j}(t, t+\Delta t)\ \ 1}{\Delta t} = \sum\limits_{i \neq j} q_{ij}, & i = j \end{cases} \tag{3.1}$$

The instantaneous transition rates form the *infinitesimal generator matrix* $Q = [q_{ij}]$

A CTMC can be mapped in to a directed graph, where the nodes represent states and the arcs represent transitions between the states labeled by their transition rates.

Definition 3.7 (The State Probability Vector): The state probability vector is a probability vector $\pi^t = \left(\pi_{s_1}^t, \pi_{s_2}^t, \cdots\right)$, where $\pi_{s_i}^t$ is the probability that the process is in state S_i at time *t*. Given the initial state probability, $\left(\pi_{s_1}^0, \pi_{s_2}^0, \cdots\right)$, the state probability vector at time *t* is calculated as:

$$\pi^t = \pi^0 P(0,t) \qquad t > 0 \tag{3.2}$$

Because of the continuity of time, Equation (3.2) cannot be easily solved; instead the following differential equation is used [16] to calculate the state probability vector from the transition rates:

$$\frac{d\pi_j(\mathrm{t})}{\mathrm{dt}} = \sum_{i \in S} q_{\mathrm{ij}}(\mathrm{t}) \pi_{\mathrm{i}}(\mathrm{t}) \mathrm{t} \tag{3.3}$$

3.1. The Steady State Analysis

For evaluating the performance of a dynamic system, it is usually desired to analyze the behavior of the system in a long period of execution. The long run dynamics of Markov chains can be studied by finding a steady state probability vector.

Definition 3.8 (The Steady State Probability Vector): The state probability vector π is steady if it does not change over time. This means that further transitions do not affect the steady state probabilities, and :

$\lim_{t \to \infty} \dfrac{d\pi(t)}{dt} = 0$, where $\pi(t)$ is the steady state probability vector. Hence from Equation (3.3):

$\sum_{i \in S} q_{ij}(t) \pi_i(t) = 0 \quad \forall j \in S$, or in vector matrix form:

$$\pi Q = 0 \tag{3.4}$$

Thus the steady state probability vector, if existing, can be uniquely determined by the solution of Equation (3.4), and the condition $\sum_{i \in S} \pi_i = 1$.

The following theorem states the sufficient conditions for the existence of a unique steady-state probability vector.

Theorem 3.1: a time-homegenous CTMC has a unique steady-state probability vector if it is irreducible and finite.

A CTMC for which a unique steady state probability vector exists is called an *ergodic* CTMC. In an ergodic CTMC, each state is reachable from any other states through one or more steps. The steady-state probability vector π for an ergodic CTMC can be found directly by solving Equation (3.4) as illustrated by the following example:

Figure 1 shows an ergodic CTMC containing five states: S_1, S_2, S_3, S_4, S_5 when each state is reachable from other states. The value associated with each arc is the instantaneous transition rate between the states. The

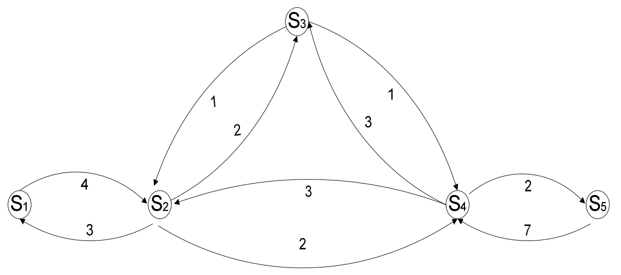

Figure 1. A simple ergodic CTMC.

infinitesimal generator matrix Q for this case would be:

$$Q = \begin{bmatrix} -4 & 4 & 0 & 0 & 0 \\ 3 & -7 & 2 & 2 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 3 & 3 & -8 & 2 \\ 0 & 0 & 0 & 7 & -7 \end{bmatrix}$$

Assuming that $\pi = [x_1, x_2, x_3, x_4, x_5]$, from Equation (3.4) we have:

$$\begin{cases} -4*x_1 + 3*x_2 = 0 \\ 4*x_1 - 7*x_2 + x_3 + 3*x_4 = 0 \\ 2*x_2 - 2*x_3 + 3*x_4 = 0 \\ 2*x_2 + x_3 - 8*x_4 + 7*x_5 = 0 \\ 2*x_4 - 7*x_5 = 0 \end{cases}$$

In addition, the summation of all $x_i \ (1 \le i \le 5)$ equals to 1, *i.e.*,

$$x_1 + x_2 + x_3 + x_4 + x_5 = 1$$

Solving the above liner system for $x_1, \cdots, x_5$, the final value of π is:

$$\pi = [7/43, 28/129, 56/129, 56/387, 16/387]$$

Notice that each value in the vector π is greater than zero which verifies the irreducible property of the CTMC, however if the some values of π are equal to 0, two possibilities exist: some states are not reachable or there are some absorbing states.

An absorbing state is a state in which no events can occur. Once a system reaches to an absorbing state, it cannot move to any other states. In other words, for an absorbing state, there are only incoming flows. Therefore, there are no outward transitions from this state. Generally, all absorbing states are failed states. A CTMC containing one or more absorbing states is called absorbing CTMC. Usually, if a system has more than one absorbing states, then all of the absorbing states can be merged into a single state.

The steady-state probability vector π for an absorbing CTMC is:

$$\pi_i = \begin{cases} 0, & \text{if } S_i \text{ is not an absorbing state} \\ 1, & \text{otherwise} \end{cases} \quad (3.5)$$

(For an absorbing CTMC, It is usually interesting to find the time to absorption. In other words, we usually want to know how long it takes for the system to step into a final absorbing state (*i.e.*, the system fails to work) from its starting state. From [16], the *mean time to absorption* (MTTA) is defined as:

$$\text{MTTA} = \sum_{i \in N} L_i(\infty) \quad (3.6)$$

where $L_i(t)$ denotes the *expected total time* spent in state S_i during the interval [0, t)

$$L_N(\infty) Q_N = -\pi_N(0) \quad (3.7)$$

Q_N is a N*N size matrix by restricting the infinitesimal generator matrix Q into non-absorbing states. As an example, assume the CTMC in **Figure 2** with an absorbing state S_5.

Here the state space Ω is partitioned into the set of absorbing states $A = \{S_5\}$ and non-absorbing states $N = \{S_1, S_2, S_3, S_4\}$. Thus the infinitesimal generator matrix Q for the non-absorbing states is reduced to:

$$Q_N = \begin{bmatrix} -4 & 4 & 0 & 0 \\ 3 & -7 & 2 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 3 & 3 & -8 \end{bmatrix}$$

Suppose $\pi_N(0) = [1, 0, 0, 0]$, which means the system starts at state S_1. Form Equation (3.7):

$$L_N(\infty) \times \begin{bmatrix} -4 & 4 & 0 & 0 \\ 3 & -7 & 2 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 3 & 3 & -8 \end{bmatrix} = -\pi_N(0)$$

Assuming $L_N(\infty) = [l_1, l_2, l_3, l_4]$, then:

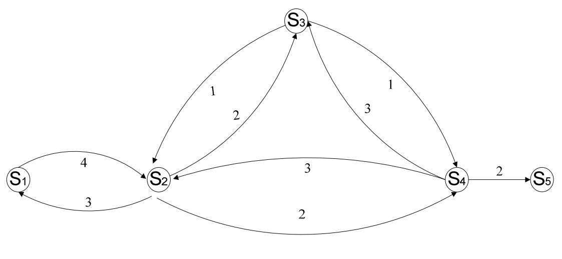

Figure 2. An absorbing CTMC.

$$\begin{cases} -4l_1 + 3l_2 = -1 \\ 4l_1 - 7l_2 + l_3 + 3l_4 = 0 \\ 2l_2 - 2l_3 + 3l_4 = 0 \\ 2l_2 + l_3 - 8l_4 = 0 \end{cases}$$

Solving the above linear system,
$L_N(\infty) = [1.0625, 1.08, 1.83, 0.5]$ and
$\text{MTTA} = \sum_{i \in N} L_i(\infty) = 17/16 + 13/12 + 11/6 + 1/2 = 4.4725.$

3.2. Markov Reward Models

Markov reward models provide a framework to integrate the performance measures with Markov chains. Computing the steady-state probability vector is not sufficient for the performance evaluation of a Markov model. A weight needs to be assigned to each state or system transition to calculate the final performance. These weights, also known as rewards, are defined according to specific system requirements and could be time cost, system availability, reliability, fault tolerance, and so forth.

Let the reward rate r_i be assigned to state S_i, then a reward $r_i t$ is accrued during a sojourn time t in state S_i. Before proceeding, some definitions are presented from [16]:

Definition 3.9 Instantaneous Reward Rate
Let $\{X(t), t \geq 0\}$ denote a homogeneous finite-state CTMC with state space Ω. The instantaneous reward rate of CTMC at time *t* is defined as:

$$Z(t) = r_X(t) \tag{3.8}$$

Based on this definition the *accumulated reward* in the finite time period [0, *t*] is:

$$Y(t) = \int_0^t Z(x)\,dx = \int_0^t r_X(x)\,dx \tag{3.9}$$

Furthermore, the expected instantaneous reward rate and the expected accumulated reward are as follow:

$$E[Z(t)] = E[r_X(t)] = \sum_{i \in \Omega} r_i \pi_i(t) \tag{3.10}$$

$$E[Y(t)] = E\left[\int_0^t r_X(x)\,dx\right] = \sum_{i \in \Omega} r_i L_i(t) \tag{3.11}$$

where $\pi(t)$ is the probability vector at time *t*, and $L_i(t)$ denotes the expected total time spent in state S_i during the interval [0, t). When $t \to \infty$ we have:

$$E[Z(\infty)] = E[r_X(\infty)] = \sum_{i \in \Omega} r_i r_i(\infty) \tag{3.12}$$

$$E[Y(\infty)] = E\left[\int_0^\infty r_X(x)\,dx\right] = \sum_{i \in \Omega} r_i L_i(\infty) \tag{3.13}$$

Especially, when the unique steady-state probability vector π of the CTMC exists, the expected instantaneous reward rate is the summation of each state reward multiplying its according steady-state probability, *i.e.*:

$$E[Z(\infty)] = E[r_X(\infty)] = \sum_{i \in \Omega} r_i \pi_i(\omega) = \sum_{i \in \Omega} r_i \pi_i \tag{3.14}$$

For an absorbing CTMC the whole system will eventually enter to an absorbing state. So for an absorbing CTMC, it is logical to compute the expected accumulated reward until absorption as follows:

$$E[Y(\infty)] = E\left[\int_0^\infty r_X(x)\,d_x\right] = \sum_{i \in N} r_i L_i(\infty) \tag{3.15}$$

where *N* is the set of non-absorbing states.

4. Performance Evaluation of Systems Modeled in Pi-Calculus

As mentioned before, the most common way to evaluate the performance of a system modeled in pi-calculus is to transfer the model to a CTMC and run the steady state analysis followed by a reward model to calculate the accumulated reward and hence the performance of the system. The major task in this regard is to assign appropriate transition rates to the Markov model. Previous literature suggests two different approaches. In the first approach, the transition rates are given explicitly for each action. This approach modifies the syntax of the pi-calculus and extends it to a stochastic pi-calculus [6-14]. The second approach [17-18] leaves the syntax of the calculus unchanged and calculates the transition rates based on an abstract cost function assigned to each action. Both of these approaches have their downsides, as previously stated in section1. In this paper, instead of a cost function, we introduce a more concrete transition rate function which estimates the transition rate for each reduction rule in pi-calculus.

4.1. Transition Rate Functions

In order to calculate the transition rates (q_{ij}) of a CTMC derived from a pi-calculus model, we define a transition rate function, *TR*, for each transition rule in higher order pi-calculus, listed in **Table 2**.

Definition 4.1 (Transition Rate Function): Let ϑ be the set of transition rules from **Table 2**. The transition rate function is defined as $TR\colon \partial \to R^+$, where $TR(\partial_i) = 1/t_i$ and t_i is the duration of the transition initiated by the reduction rule ∂_1.

In what follows the transition rate functions are defined for the higher order pi-calculus reduction rules.

- **TAU:** $\dfrac{-}{\tau.P \xrightarrow{\tau} P}$

$$\text{TR}\,\frac{-}{\tau_i.P \xrightarrow{\tau_i} P} = \frac{1}{\lambda_i}$$

Here τ_i is an internal action. Different internal actions may have different execution times, so λ_i represents the execution time for action τ_i.

- **SUM:** $\dfrac{P \xrightarrow{\tau} P'}{P+Q \xrightarrow{\tau} P'}$

Similar to TAU, $TR\left(\dfrac{P \xrightarrow{\tau} P'}{P+Q \xrightarrow{\tau} P'}\right)$

$$= TR\left(P \xrightarrow{\tau_i} P'\right) = \frac{1}{\lambda_i}$$

- **PAR:** $\dfrac{P \xrightarrow{\tau} P'}{P|Q \xrightarrow{\tau} P'|Q}$

$$TR\left(\frac{P \xrightarrow{\tau} P'}{P|Q \xrightarrow{\tau} P'|Q}\right) = TR\left(P \xrightarrow{\tau_i} P'\right) = \frac{1}{\lambda_i}$$

- **MATCH:** $\dfrac{P \xrightarrow{\tau} P'}{[x=x]P \xrightarrow{\tau} P'}$

$$TR\left(\frac{P \xrightarrow{\tau_i} P'}{[x=x]P \xrightarrow{\tau_i} P'}\right) =$$

$TR\left([x=x]P \xrightarrow{\tau_i} P'\right)$ = 1/(time($[x=x]$) + time $\left(P \xrightarrow{\tau_i} P'\right)) \approx \dfrac{1}{size(x)+\lambda_i}$

The time needed for the matching operation, depends on the size of the data to be compared, size(*x*).

- **COM-NAME:** $\dfrac{P \xrightarrow{\bar{x}\tilde{y}} P' \qquad Q \xrightarrow{x(\tilde{z})} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{\tilde{y}/\tilde{z}\}}$

Assuming that the network connection is perfect and no errors occur during sending and receiving messages, we have:

$$TR\left(\frac{P \xrightarrow{\bar{x}\tilde{y}} P' \qquad Q \xrightarrow{x(\tilde{z})} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{\tilde{y}/\tilde{z}\}}\right) = \frac{1}{t_{startup} + \dfrac{size(\tilde{y})}{bw(x)} * (k+1) + t_h * k}$$

where $bw(x)$ is the bandwidth of channel *x*, *i.e.*, the total amount of information that can be transmitted over channel x in a given time. When a large number of communication channels are using the same network, they have to share the available bandwidth. $t_{startup}$ is the startup time, the time required to handle a message at the sending node. This includes the time to prepare the message (adding header, trailer, and error correction information), the time to execute the routing algorithm, and the time to establish an interface between the local node and the router. This delay is incurred only once for a single message transfer [20]. *k* is the total hops that a message traverses (not counting the sender and the receiver nodes). t_h, called Per-hop time, is the time cost at each network node. After a message leaves a node, it takes a finite amount of time to reach the next node in its path. The time taken by the header of a message to travel between two directly-connected nodes in the network is called the per-hop time. It is also known as node latency. The per-hop time is directly related to the latency within the routing switch for determining which output buffer or channel the message should be forwarded to $size(\tilde{y})$ represents the size of the message $\tilde{y}$. In current parallel computers, the per-hop time, t_h, is quite small. For most parallel algorithms, it is less than $size(\tilde{y})/bw(x)$ even for small values of m and thus it can be ignored.

As an example, suppose the sender begin to send 200K $(size(\tilde{y}))$ message to the recipient in **Figure 3**. The bandwidth of the network is 100 K/second and the sender startup time is 1.5 second. At each hop, the latency (t_h) is 1 second. So the total transfer time *t* is calculated as:

$$t = t_{startup} + \frac{size(\tilde{y})}{bw(x)} * (k+2) + t_h * k = 1.5 + (200/100) * (3+1) + 1 * 3 = 12.5 \text{ second}$$

And its corresponding transition rate is 1/*t* = 1/14.5 = 0.08

- CLOSE-NAME:

$$\frac{P \xrightarrow{\bar{x}(\tilde{w})} P' \qquad Q \xrightarrow{x(\tilde{w})} Q'}{P|Q \xrightarrow{\tau} (w)(P'|Q')}$$

$$TR\left(\frac{P \xrightarrow{\bar{x}(\tilde{w})} P' \qquad Q \xrightarrow{x(\tilde{w})} Q'}{P|Q \xrightarrow{\tau} (w)(P'|Q')}\right) = \frac{1}{t_{startup} + Check(\tilde{w}) + \dfrac{size(\tilde{y})}{bw(x)} * (k+1) + t_h * k}$$

$Check(\tilde{w})$ is the time needed to perform a name check to verify that $\tilde{w}$ does not occur free in Q. $Check(\tilde{w})$ is in the order of the number of free names in *Q*.

- COM-PROCESS:

$$\frac{P \xrightarrow{(v\tilde{y})\bar{x}\langle\tilde{K}\rangle} P' \qquad Q \xrightarrow{x\langle\tilde{K}\rangle} Q'}{P|Q \xrightarrow{\tau} v\tilde{y}(P'|Q')}$$

$$TR\left(\frac{P \xrightarrow{(v\tilde{y})\bar{x}\langle\tilde{K}\rangle} P' \qquad Q \xrightarrow{x\langle\tilde{K}\rangle} Q'}{P|Q \xrightarrow{\tau} v\tilde{y}(P'|Q')}\right)$$

$$= \frac{1}{Marshalling(\tilde{K}) + t_{startup} + \frac{size(\tilde{K}code + \tilde{K}data + \tilde{K}state)}{bw(x)} * (k+1) + t_h * k + Unmarsh(\tilde{K})}$$

When a process is moved to a remote computer, its code, data, and, sometimes, its state of execution are transferred so that it can resume its execution after migration [21]. $\tilde{K}code$, $\tilde{K}data$, $\tilde{K}state$ are the code, data and the state of process $\tilde{K}$, respectively. $Marshalling\left(\tilde{K}\right)$ is the time needed to convert $\tilde{K}$ into a standard format (byte-stream) before it is transmitted over the network. $Unmarshalling\left(\tilde{K}\right)$ is the reverse process of converting the byte-stream to process $\tilde{K}$, after migration.

With the transition rate functions defined above, a system modeled in pi-calculus can be transformed into a CTMC and hence its performance measures are computed by following the standard numerical methods of CTMC, discussed in Section 3. Generally, the steps needed to be taken for CTMC-based performance evaluation of a system described in pi-calculus or its extensions are summarized as follow:

1) Forming the state diagram of the system. To do so, we consider the initial pi-calculus specification to represent the initial state of the system. From there every reduction applied to the specification would produce a next possible state. Similarly the state transition diagram is generated by applying all possible reductions.

2) Using the state transition rate functions to calculate the transition rates for each pair of states. We define the transition rate, q_{ij}, between state S_i and S_j, based on the transition rate function:

$$q_{ij} = \begin{cases} TR(\partial), \text{ if } S_j \text{ is derived inmediately from} \\ \qquad S_i \text{ by applying the reduction rule } \partial \\ \sum_j q_{ij}, \; i = j \\ 0, \quad \text{otherwise} \end{cases} \tag{4.1}$$

3) By associating each rate with its appropriate transition in the state diagram, the CTMC diagram would be formed and the infinitesimal matrix is generated. Note that the transition rates are independent of the time (It only depends on the transition rule); hence the corresponding CTMC is time-homogenous. To ensure that the resulting transition system conforms to the memory-less property, we assume that the sojourn time of a state is exponentially distributed with parameter $1/\sum_j q_{ij}$ [16].

That is:

$$prob\left(sojourn\left(S_i\right) \le t\right) = 1 - e^{\lambda_i t} \tag{4.2}$$

where $\lambda_i = 1/\sum_j q_{ij}$

4) If the resulting CTMC is an ergodic Markov Chain, then a unique steady state probability vector exists and is found by solving Equation (3.4). If a state based Markov reward model is used, then the final performance value of the system would be equal to: $\sum_{i=1}^{n} r_i \pi_i$, where r_i and π_i are the reward rate and the steady probability for state S_i, respectively.

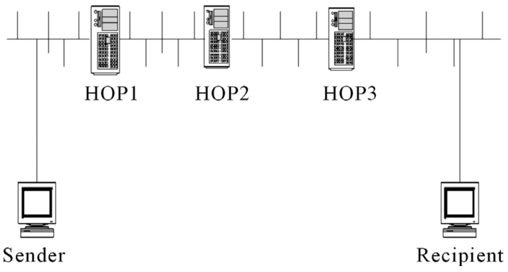

Figure 3. Message sending and receiving across the network [20].

5. Conclusion and Future Work

Although the pi-calculus family of formal languages is vastly utilized, it does not provide any native theoretical mechanisms for performance evaluation of different design approaches for a particular system. This article presented a Markov-based performance evaluation methodology for systems specified in pi-calculus. Such methodology can be applied at the design stage to assess the performance of a system prior to its implementation and introduce a benchmark to compare different design schemes against one another.

The previous works [22-24] on integrating performance measures with process algebra have either introduced additional computational overhead by extending the calculus to stochastic pi-calculus or proposed cost functions that are too abstract. We defined concrete transition rate functions with necessary features for real world applications. Bearing in mind that the performance of a multi-process system mainly relies on the network communication cost, our transition rate functions mostly revolves around the time performance measures for sending and receiving processes. In addition, we also accounted for the overhead of process mobility in our transition functions.

The future studies should mainly focus on the scalability of the methodology to large complex systems with huge number of states and transition rates. Many approaches are proposed in literature to address the problem of state explosion in Markov chains [25-27].The incorporation of these approaches with the performance evaluation of pi-calculus models remains as a future work.

REFERENCES

[1] R. Milner, J. Parrow and D. Walker, "A Calculus of Mobile Processes—Part I and II," LFCS Report 89-85, University of Edinburgh, Edinburgh, 1989.

[2] R. Milner, "Communicating and Mobile Systems: The π -Calculus," Cambridge University Press, Cambridge, 1999.

[3] R. Milner, "The Polyadic Pi-Calculus: A Tutorial," Technical Report ECSLFCS -91-180, Computer Science Department, University of Edinburgh, Edinburgh, 1991.

[4] D. Sangiorgi, "The π -Calculus: A Theory of Mobile Processes," Cambridge University Press, Cambridge, 2001.

[5] D. Sangiorgi: "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms," Ph.D. Thesis, University of Edinburgh, Edinburgh, 1993.

[6] C. Priami, "Stochastic π -Calculus," *Computer Journal*, Vol. 38, 1995, pp. 578-589. doi:10.1093/comjnl/38.7.578

[7] N. GÖtz, U. Herzog and M. Rettelbach, "TIPP-A Language for Timed Processes and Performance Evaluation," Technical Report 4/92. IMMD VII, University of Erlangen-Nurnberg, Erlangen, 1992.

[8] J. Hillston, "A Compositional Approach to Performance Modelling," Ph.D. Thesis, University of Edinburgh, Edinburgh, 1994.

[9] M. Bernardo, L. Donatiello and R. Gorrieri, "MPA: A Stochastic Process Algebra," Technial Report UBLCS-94-10, University of Bologna, Bologna, 1994.

[10] P. Buchholz, "On a Markovian Process Algebra," Techinal Report Informatik IV, University of Dortmund, Dortmund 1994.

[11] L. de Alfaro, "Stochastic Transition Systems," *Proceedings of Ninth International Conference on Concurrency Theory* (*CONCUR*'98), Vol. 1477, 1998, pp. 423-438. doi:10.1007/BFb0055639

[12] J. Markovski and E. P. de Vink, "Performance Evaluation of Distributed Systems Based on a Discrete Real- and Stochastic-Time Process Algebra," *Fundamenta Informaticae*, Vol. 95, No. 1, October 2009, pp. 157-186.

[13] A. Clark, S. Gilmore, J. Hillston and M. Tribastone, "Stochastic Process Algebras," *SFM*'07 *Proceedings of the* 7*th International Conference on Formal Methods for Performance Evaluation*, 2007, pp 132-179.

[14] P. R. D'Argenio and J. Katoen, "A Theory of Stochastic Systems. Part II: Process Algebra," *Information and Computation*, Vol. 203, No. 1, November 2005, pp. 39-74. doi:10.1016/j.ic.2005.07.002

[15] S. M. Ross, "Stochastic Processes," 2nd Edition, Wiley, New York, 1996.

[16] G. Bolch, S. Greiner, H. De Meer and K. S. Trivedi, "Queuing Networks and Markov Chains: Modelling and Performance Evaluation with Computer Science Applications," Wiley, New York, 1998

[17] C. Nottegar, C. Priami and P. Degano, "Performance Evaluation of Mobile Processes Via Abstract Machines," *IEEE Transactions on Software Engineering*, Vol. 27, No. 10, 2001, pp. 867-889. doi:10.1109/32.962559

[18] F. Logozzo, "Pi-Calculus as a Rapid Prototype Language For Performance Evaluation," *Proceedings of the ICLP* 2001 *Workshop on Specification, Analysis and Validation for Emerging Technologies in Computational Logic* (*SAVE* 2001), 2001.

[19] S. Rahimi, M. Cobb, D. Ali, M. Paprzycki and F. Petry, "A Knowledge-Based Multi-Agent System for Geospatial Data Conflation," *Journal of Geographic Information and Decision Analysis*, Vol. 6, No. 2, 2002, pp. 67-81.

[20] A. Grama, G. Karypis, V. Kumar and A Gupta, "An Introduction to Parallel Computing: Design and Analysis of Algorithms," 2nd Edition, Addison Wesley, Reading, 2003.

[21] W. R. Cockayne and M. Zyda, "Mobile Agents," Manning Publications Company, Greenwich, 1998.

[22] H. Samet, "The Design and Analysis of Spatial Data Structures," Addison-Wesley, Reading, 1989.

[23] S. Rahimi, "Using Api-Calculus for Formal Modeling of SDIAgent: A Multi-Agent Distributed Geospatial Data Integration," *Journal of Geographic Information and Decision Analysis*, Vol. 7, No. 2, 2003, pp. 132-149.

[24] S. Rahimi, J. Bjursell, M. Paprzycki, M. Cobb and D. Ali, "Performance Evaluation of SDIAGENT, a Multi-Agent System for Distributed Fuzzy Geospatial Data Conflation," *Information Sciences*, Vol. 176, No. 9, 2006. doi:10.1016/j.ins.2005.07.009

[25] S. Derisavi, P. Kemper and W. H. Sanders, "Symbolic State-Space Exploration and Numerical Analysis of State-Sharing Composed Models," *Linear Algebra and Its Applications*, Vol. 386, July 2004, pp. 137-166. doi:10.1016/j.laa.2004.01.006

[26] S. Derisavi, H. Hermanns and W. H. Sanders, "Optimal State-Space Lumping in Markov Chains," *Information Processing Letters*, Vol. 87, No. 6, 2003, pp. 309-315. doi:10.1016/S0020-0190(03)00343-0

[27] P. Buchholz, "Efficient Computation of Equivalent and Reduced Representations for Stochastic Automata," *International Journal of Computer Systems Science & Engineering*, Vol. 15, No. 2, March 2000, pp. 93-103.