Shared Transaction Markov Chains for Fluid Analysis of Massively Parallel Systems

Richard Hayden Jeremy T. Bradley
Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, United Kingdom.
{rh,jb}@doc.ic.ac.uk

***Abstract*—We introduce a low-level performance modelling formalism, Shared Transaction Markov Chains (STMCs), specifically designed for the capture and analysis of massively parallel stochastic systems through fluid techniques. We introduce the notion of a *shared transaction* between concurrently running Markov chains which allows a multi-phase synchronisation to accurately represent complex cooperation between modelling components in a compositional manner. We demonstrate the new modelling formalism on four distinct models and show how fluid analysis may be performed, with results, where appropriate. Our contribution is that this is the first such system tailored to the fluid performance analysis of transaction-based systems as found in computing applications such as peer-to-peer networks, web architectures and Publish-Subscribe networks. The second contribution is that STMCs permit composed phase-type distributed synchronisation which is more useful from a transaction modelling perspective.**

I. Introduction

Over the last two decades, performance modelling using high-level formalisms such as Generalised stochastic Petri nets (GSPNs) [1], stochastic process algebras (SPAs) [2], [3], [4], stochastic automata networks [5] and stochastic activity networks [6] has enjoyed considerable uptake and use in many diverse application areas. It is worth taking a moment to understand why these techniques have been as successful as they have. Several fundamental reasons can be cited: ease-of-use, easy problem visualisation, high quality tool support and a straight-forward correspondence with the behavioural model of the application.

However, the success of such formalisms has, at the same time, emphasised the shortcomings of the underlying analysis techniques. Most of the common formalisms deploy an underlying continuous-time Markov chain (CTMC) stochastic model, as a means of capturing both performance and concurrency easily. Initially, steady-state analysis was used to generate measures such as mean utilisation and throughput within the process algebra or Petri net model.

However, it is hard to avoid generating massive state spaces when constructing realistic SPA or GSPN models and it quickly became clear that the available linear algebra tools could not analyse the massive state spaces being generated. Computer performance improved and parallelisation gave access to steady state analysis of 10^8 state models and beyond [7]. However, required performance measures became more computationally intensive and the application of transient analysis [8] and response-time analysis [9] to these higher-level formalisms has further impeded the ability to analyse truly large models. Recently, fluid techniques have been developed to analyse massive state spaces generated by exploiting the parallelism present in many models. This fluid analysis has been applied in both stochastic process algebra [10], [11], [12] and stochastic Petri net [13] formalisms and is restricted by the size of the local component description or net, rather than the global state space.

Another problem has been the ability of some of the formalisms to capture the performance details of applications, especially in being able to specify the exact stochastic duration of compositional, N-way synchronisation between components. Formalisms such as Layered Queueing Networks [14] and IMC [4] countered this with built-in multi-phase behaviour within the synchronisation [15]. Other formalisms such as semi-Markov SPNs [16] and semi-Markov SPAs [17] used a more general underlying stochastic model, but this restricted the ability of the modeller to cope with concurrency; a critical feature of most modern-day applications.

So the problem tackled in this paper combines these two issues – how can we tackle massive state spaces, by fluid analysis where appropriate, whilst maintaining an accurate performance model of the cooperation between stochastic components. Our solution is to develop a low-level compositional formalism which, based on the experience with the stochastic process algebra, PEPA [18], [10], [12], will yield an intuitive fluid analysis for a large class of the models which can be represented by the formalism. Simultaneously, we permit multi-phase synchronisation distributed between the cooperating components, to allow both for a more general timing model of the synchronisation and a structurally richer set of possible interactions between components.

In particular, we develop the notion of a *shared transaction* between components. A key feature of this construction is that it allows one component to *begin* a shared transaction with *any* of a large number of homogeneous components, but then to continue the remaining phases of the synchronisation with the one particular partner chosen at the start. This is difficult to represent in existing compositional action-based formalisms such as PEPA, especially, when we are dealing with very large component groups, as is the case with fluid analysis. The reason for this difficulty is that the partner chosen at the start must be explicitly encoded into the enabled action

types for the remaining phases of the synchronisation, which leads to unnecessarily large local state spaces for the individual components.

At this stage we are not developing a full behavioural modelling formalism as there are many existing candidates which do an excellent job of capturing model structure, interaction and abstraction. Rather we introduce a performance-level data structure which represents cooperating Markov chain components and has just enough embedded behavioural detail to permit multi-phase synchronisation between them.

This will allow us or others, in later work, to create translations from existing formalisms where fluid techniques have already been exploited (PEPA [10], [12], SPNs [13], Stochastic π-calculus [19], sCCP [20]) and also permit a fluid analysis route in formalisms where more general synchronisation is already possible in some form (LQNs and IMC).

For now though, our contribution is that we present the first compositional performance formalism, which allows multi-phase synchronisation between components, and is aimed at directly facilitating fluid analysis. To enable fluid analysis, we consider homogeneous groups of parallel Markov chains, which we call *agents* and allow an STMC configuration to include information about the size of these groups thus facilitating scalability analysis.

We permit the use of immediate transitions as they have been shown to be very useful in, for example, measurement specification and queueing models. Furthermore, by allowing the use of different selection policies between Markov chains that can transact with more than one possible partner, we allow the expression of bespoke selection distributions. We note that most of the formalisms discussed above use a solely uniform selection policy.

In the next section (Section II), we define what constitutes an STMC model and then in Section III, we introduce an intuitive graphical representation and present a simple example. Section IV defines the intended stochastic behaviour of an STMC model and finally, Section V introduces several more examples, along with fluid analysis results where that is currently possible.

II. STMC DEFINITION

In this section, we define what constitutes a Shared Transaction Markov Chain (STMC) model.

Each STMC model consists of two components, the *specification* and the *configuration*. The specification is the main part of an STMC and it defines the stochastic behaviour of all of the types of component and how they can interact. The configuration specifies the numbers of each type of component present, and their initial states.

This separation is anticipated to be useful in many modelling scenarios. For example, when modelling for scalability analysis, the sizes of the population of certain component types may be altered jointly to satisfy a certain performance requirement. This is also helpful in stating stochastic limit results, which are often specified in terms of sequences of models with larger and larger component populations.

Specifications are defined in the next section and configurations in the proceeding section.

A. STMC model specifications

A Shared Transaction Markov Chain specification is an 8-tuple, $(N_A, A, N_T, T, R, P, S, V)$. N_A is the set of *agent type names*, which name and are in one-to-one correspondence with *agents*, which make up the set A. N_T is the set of *transaction names*, which similarly name and are in one-to-one correspondence with *transaction specifications*, which make up the set T. R is the set of *role names*, P is the *selection policy*, S is the set of *signal names* and V is a set of formal variables. All of these sets must be finite.

The elements of N^A, N^T, R and S can be considered formally just as string labels describing the intended purpose of the agent type, transaction, role or signal, respectively. Selection policies can be treated simply as formal objects without any further structure for the moment until Section IV-C, where they are discussed.

The foremost components of a model are the agent types and transaction specifications. The following two paragraphs introduce these concepts and their relationship briefly, before they are properly introduced in the next two sections.

Agents are the atomic components within a model. They are best viewed as individual Markov chains, which run in parallel together, until they reach states in which interactions, in the form of shared transactions between agents, become possible. An *agent type* specifies the stochastic behaviour of a particular type of agent. As mentioned above, the STMC configuration will specify how many 'copies' of a given agent type there are in a model. In this way, we should view agent types as templates for a number of agents who behave identically.

Transaction specifications define the possible interactions between individual agents of each different type. For each transaction, they simply specify the roles which each must be fulfilled by an agent for the shared transaction to go ahead.

1) Transaction specifications: More formally, a transaction specification is a tuple $(t, \mathcal{R})$, where $t \in N_T$ is the name of the transaction being specified and $\mathcal{R} \in \mathcal{P}(R)$ is the set of roles required for the transaction.[1] The interpretation is that for this transaction to go ahead, there must be an agent offering each of the required roles.

For example, a simple interaction might consist of clients who download from servers. The transaction might be called *download* with roles, *downloader* and *uploader*, offered by the clients and servers, respectively.

We will see in more detail what it means for an agent to offer a role in the next section.

2) Agent types: Formally, an agent type is a 5-tuple, $(a, X, D, T^>, T^<)$, where $a \in N_A$ is simply this agent type's unique name and X is the finite set of *states* of the agent type (represented in our examples by a set of strings giving textually the meaning of each state). D specifies *transitions*,

[1] We use the notation $\mathcal{P}(X)$ throughout to mean the *powerset* of X, that is, the set of all subsets of X.

that is, how agents of this type may move between their states. Formally, D is a finite subset of:

$$\{(x,\, y,\, k,\, r,\, S_g,\, S_t)\ : x,\, y \in X,\, k \in \{\mathbf{t},\, \mathbf{i}\},\, r \in \mathbb{R}^+,\\ S_g,\, S_t \in \mathcal{P}(V \times S)\}$$

where:

- x and y specify the start and end states of the transition, respectively;
- $k = \mathbf{t}$ for *timed transitions* or $k = \mathbf{i}$ for *immediate transitions*;
- r is a non-negative rate parameter for timed transitions or relative probabilistic weighting for immediate transitions;
- S_g and S_t, specify the *guarded signals* and the *transmitted signals*, respectively.

a) Signals: Signals are only meaningful when an agent is engaged in a shared transaction with one or more other agents. Signals provide the mechanism through which these agents engaged in transactions together may communicate. This allows the precise stochastic timing of the shared transaction to be constructed in a compositional manner by all parties involved in the transaction. As we will see later in this section and in more detail in Section IV, the other partners in a shared transaction may be referenced by an agent using one of the variables in the set V.

The transmitted signals, S_t, specify which signals are transmitted to which transaction partners when the transition fires. So if $(v,\, s) \in S_t$, this represents the transmission of the signal s to the partner agent referenced within the shared transaction by the variable v.

On the other hand, the guarded signals, S_g, specify signals which must have been received from certain transaction partners before the transition is enabled. Thus if $(v,\, s) \in S_g$, this transition is only enabled when the agent has received the signal s from the partner agent referenced within the shared transaction by the variable v.

b) Transaction entry and end points: $T^>$ is the set of *offered transactions*. For each possible agent state, the offered transactions specify all of the shared transactions into which the agent can enter when it is in that state and the role the agent offers to play in each. They also specify the variables which will be used by the agent to reference the other agents playing the other roles required by the transaction.

More formally, $T^>$ is a finite subset of:

$$\{(x,\, t,\, o,\, y,\, B)\ : x \in X,\, t \in N_T,\, o \in R,\, y \in X,\\ B \in \mathcal{P}(V \times R)\}$$

- x specifies the state in which an agent of this type offers to play the role, o, in a transaction of type, t;
- y is the state into which agents of this type will move immediately if this shared transaction begins. We call this implicit immediate transition a *transactional immediate transition*;
- Each element, $(v,\, r) \in B$, specifies that the variable, v, will be used to reference the agent playing the role, r, should this shared transaction go ahead.

Every state offering one or more shared transactions is called a *transaction entry point*. To ease our notation, we do not allow two entries, $(x,\, t,\, o,\, y,\, B)$ and $(x,\, t,\, o,\, y',\, B') \in T^>$, that is, differing in at most the final two places, since it is possible to add extra states and transitions to agent types to achieve the same meaning.

Finally, $T^<$ specifies *transaction end points*. These define where transaction interactions with other agents come to an end. Formally, $T^<$ is a finite subset of $\mathcal{P}(D \times V)$. Each element $(d,\, v) \in T^<$ specifies that when the transition, d, fires, this agent's transaction interaction with the agent referenced by the variable, v, ends.

B. STMC model configurations

Consider an STMC model specification, $(N_A,\, A,\, N_T,\, T,\, R,\, P,\, S,\, V)$. For a given agent type name $a \in N_A$, define $X(a)$ to be the set of states of the agent type corresponding to a, which was introduced in Section II-A2. Let $\mathbf{X} := \cup_{a \in N_A} X(a)$.

Then an STMC model configuration for this specification is simply a function $\mathbf{C} : N_A \times \mathbf{X} \to \mathbb{N}$ specifying the number of each agent type, which start in each state, in the model. For this to be meaningful, we require that if $x \notin X(a)$, then $\mathbf{C}(a,\, x) = 0$ for all $a \in N_A$, i.e. each instance of each agent starts in one of its own states. Write also $\mathbf{C}(a) := \sum_{x \in X(a)} \mathbf{C}(a,\, x)$ for the total number of a given agent type in the model.

III. Graphical notation

In this section, we give a straightforward and intuitive graphical representation for agent types, illustrated by a simple example. The notation is also used in Section V, where we present further examples.

A summary of this graphical syntax is given in Table I. For each agent type, $(a,\, X,\, D,\, T^>,\, T^<) \in A$, we construct a labelled and directed graph whose nodes correspond to the elements of X and whose arcs correspond to the transitions specified by D. Graph nodes corresponding to transaction entry points are drawn as rectangles, whereas circles are used for other nodes. Within transaction entry points, we write the details of the offered transactions, which is also shown in Table I.

Arcs corresponding to immediate transitions are drawn as dashed red lines, and for timed transitions, we use solid black lines. The other information about each transition, such as rates or weightings, guarded signals and transmitted signals is specified by annotating the respective arc, as shown in Table I. Transactional immediate transitions are distinguished by annotating the respective arc with a circle. Transaction end points are distinguished with a square and the variables referencing the agents with whom the transaction is being ended are listed next to it.

A. Example

We now use the graphical representation just introduced to present a simple example of an STMC model specification consisting of two agent types, representing clients and servers.

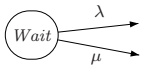
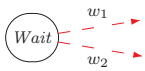
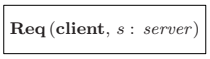
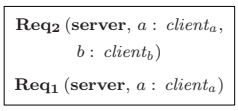
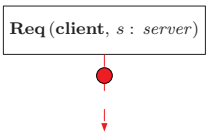
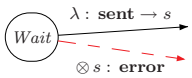
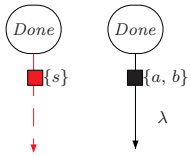
	Node representing the agent state *Wait*, which is not a transaction entry point.
	A node representing a state with two timed transitions with rates, λ and μ, respectively.
	A node representing a state with two immediate transitions with relative probabilistic weightings, w_1 and w_2, respectively.
	Node representing an agent state which is a transaction entry point with one offered transaction, **Req**. This agent offers to play the role **client** and whichever other agent plays the role *server* will be bound to the variable s if this shared transaction goes ahead.
	Node representing an agent state which is a transaction entry point with two offered transactions. For the first, **Req₂**, this agent offers to play the role **server** and the agents which play the roles $client_a$ and $client_b$ will be bound to the variables a and b, respectively. In the second, **Req₁**, this agent again offers the role **server** and the agent which plays the role $client_a$ will be bound to the variable a.
	Transactional immediate transition which fires if and when the (only) transaction begins. If the state corresponding to this node offered more than one transaction, extra labelling would be required. It would be necessary to identify the transaction which corresponds to each transition.
	A node corresponding to a state with a timed transition of rate λ. It transmits a **sent** signal when it fires to another agent in an ongoing shared transaction referenced by s. It also has an immediate transition which is only enabled if an **error** signal has been received from the same agent.
	Transaction end points. The left-hand one is on an arc corresponding to an immediate transition. It completes the interaction with the agent referenced by s. The right-hand one is on an arc corresponding to a timed transition and ends the interaction with the agents referenced by a and b.

TABLE I: Graphical summary of STMC syntax.

Clients must first enter into a shared transaction with a server in order to have some data processed. Following this, clients then complete some task independently, and the servers must perform a reset operation, also independently.

The graphical representation of the two agent types in the model, **SimpleClient** and **SimpleServer**, is given in Figure 1. The interaction of the clients and servers is specified by the transaction called **Proc**, which requires an agent to fulfil each of the two transaction role names, *client* and *server*. These represent the client and server role in the processing task, respectively. Formally, the corresponding transaction specification is (**Proc**, {*client*, *server*}).

We require two signal names, **sent** and **processed**. **sent** is used by a client to indicate to the server that it has transmitted its initial request data, and **processed** is used by a server

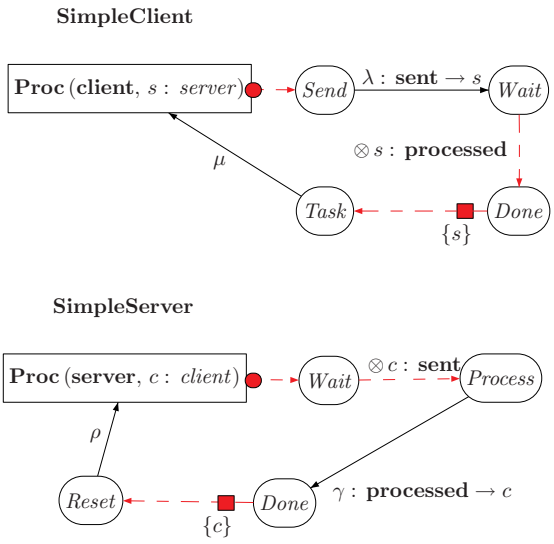


Fig. 1: Agent types for simple client/server example.

to indicate to the client that it has completed the required processing request. The formal variable names used by this model are just c and s, for the server to refer to the client in their transaction and vice-versa, respectively.

The formal definition of the entire model specification can be found in the appendix of the extended technical report [21]. We will return to this example at the start of Section V.

IV. Stochastic Semantics

In this section, we give the stochastic semantics of an STMC model. We wish to associate a stochastic process with each model, which will keep track of the state of every agent instance within the model. We intend that this process will be a continuous-time Markov chain (CTMC) on an appropriate finite global state space.

The global state space will need to keep track of the state of each agent within the model and the partner agents to which variables are bound for any ongoing shared transactions. Furthermore, the set of signals received by each agent from transaction partners will also need to be included in the global state space.

A. Global state space

Let the model under consideration be defined by the model specification, $(N_A, A, N_T, T, R, P, S, V)$ and configuration, $\mathbf{C}$. Assume each agent in the model of type $a \in N_A$ has a unique identifier and let $\mathcal{I}_a$ be the set of all such identifiers. Then write for the disjoint union, $\mathcal{I} := \uplus_{a \in N_A} \mathcal{I}_a$, the set of unique identifiers of all agents in the model. For $i \in \mathcal{I}$, let $x_0(i)$ be the initial state of agent i. In order to satisfy the configuration, we require that for each $a \in N_A$ and $x \in X(a)$, the correct number of agents start in state x:

$$|\{i \in \mathcal{I}_a \,:\, x_0(i) = x\}| = \mathbf{C}(a, x)$$

$X(a)$	Set of possible states agents of type $a \in N_A$ can be in
$D(a)$	Set of possible transitions for agents of type $a \in N_A$
$T^<(a)$	Set of transaction end points for agents of type $a \in N_A$
$T^>(a)$	Set of offered transactions for agents of type $a \in N_A$

TABLE II: Notation for components of the specification of an agent type.

Some straightforward notation for the aspects of the specification of an agent type, as defined in Section II, is required before proceeding further. We present this in Table II.

Formally, we then define the global state space, $\mathbf{G}$, to be the set of all functions from $\mathcal{I}$ to the set:

$$\mathbf{X} \times \mathcal{P}(V \times S) \times \mathcal{P}(V \times \mathcal{I})$$

Let $g \in \mathbf{G}$, then for each agent $i \in \mathcal{I}$, $g(i) = (x(i), R(i), V(i))$ is a 3-tuple. The first element of $g(i)$ keeps track of the state agent i is currently in, thus if $i \in \mathcal{I}_a$, $x(i) \in X(a)$. The second element keeps track of the received signals: each pair, $(v, s) \in R(i)$ specifies that signal s has been received from a partner agent in a shared transaction bound to the variable v. The third element keeps track of the agents to which variables are bound: each pair $(v, j) \in V(i)$ specifies that variable v is bound to agent j as part of an ongoing shared transaction. We note that all states in the global state space will not always be reachable.

The initial state of the global state space is defined as $i \in \mathcal{I} \to (x_0(i), \emptyset, \emptyset)$, representing each agent beginning in its initial state as specified by the configuration. Since the last two elements of the initial state are $\emptyset$, models cannot begin in the middle of shared transactions.[2]

1) Transitions in the global state space: We will specify the stochastic semantics of STMC models by defining, for a given global state, $g \in \mathbf{G}$, the possible transitions from it to other states in the global state space.

We will define both timed and immediate transitions. Timed transitions will be assigned a non-negative real rate parameter, which is the parameter of an independent exponential random variable. Races between only timed transitions are then stochastically determined in the usual manner by firing the transition corresponding to the exponential random variable with smallest value, which is the state holding time.

Immediate transitions will each be assigned a probability value (that is, a real value in $[0, 1]$). These will sum to one over all outgoing immediate transitions for the current global state. Races between only immediate transitions are then stochastically determined in the obvious manner. Races between immediate and timed transitions are resolved by ignoring the timed transitions. Any states with one or more immediate transitions is a vanishing state, that is, it has zero state holding time.

[2]This limitation could be overcome by augmenting the definition of STMC model configurations to support extra information specifying ongoing shared transactions.

For any $g \in \mathbf{G}$, we consider two possible cases: either there are shared transactions which are ready to begin between two or more agents (in which case we call g a *transaction start state*), or there are not (g is a *simple state*). If g is a transaction start state, we adopt the convention of starting the next shared transaction before allowing any other transitions to proceed. We will define formally what it means for a shared transaction to be ready to begin and exactly how this case is dealt with shortly (Section IV-C). For now, we begin with the case of simple states.

B. Simple states

Let $g = i \in \mathcal{I} \to (x(i), R(i), V(i))$ be a simple state in the global state space. To define the possible transitions from g to other global states, we consider for each agent type, $a \in N_A$, every individual agent of this type, $j \in \mathcal{I}_a$, in turn. In particular, we are interested in the local transitions possible for agent j while it is in its current state, $x(j)$. Recall that the local transitions possible for agents of type a are defined by the set $D(a)$. In particular, we wish to consider every element $d = (x, y, k, r, S_g, S_t) \in D(a)$, which has $x = x(j)$, that is, which defines a transition out of the current state of the agent j.

For each such local transition, it is enabled if and only if we have received all of the signals which guard it, that is, $S_g \subseteq R(j)$. If and only if this is the case, we add a transition in the global state space corresponding to this local transition.

The end state for this global transition is $g_e = i \in \mathcal{I} \to (x_e(i), R_e(i), V_e(i))$, where g_e is the same as g except for:

- $x_e(j) = y$ because agent j has moved to state y;
- If signals are transmitted when d fires, that is, if $S_t \neq \emptyset$, we must register their transmission in the end state, g_e. Specifically, for each $(v, s) \in S_t$, we need to establish which agent v currently refers to. To do this, we must look it up in the variable bindings for the current state, $V(j)$. That is, we seek the $b \in \mathcal{I}$, for which $(v, b) \in V(j)$. Then we must find the variable that agent b is using to refer to j. This is the $z \in V$, for which $(z, j) \in V(b)$. We can then form the pair (z, s), which we add to $R_e(b)$. This means that in the global state, g_e, agent b sees the receipt of the signal, s, from agent, j, which it is referring to through variable, z.
- If any transactions come to an end when the local transition under consideration, d, fires, we must remove the corresponding variable bindings. That is, we must remove the bindings for variables, which are specified by a transaction end point on the local transition, d. Formally, consider the set, say V', of all $v \in V$, such that $(d, v) \in \mathcal{T}^<(a)$, that is, each variable, which is specified by a transaction end point on the current local transition, d. Then we define:

  $$V_e(j) = V(j) \setminus \{(v', \cdot) \in V(j) : v' \in V'\}$$

  That is, we remove any bindings for variables in V' in the end state, g_e.

- We must also clear any signals received from agents referenced by variables which are specified by a transaction end point on the current local transition, d. This is so that variables can be re-used. Furthermore, we also wish to clear any signals observed by a guard on the current local transition so that signals can be re-used within a shared transaction. Therefore, we define:

$$R_e(j) = R(j) \setminus (S_g \cup \{(v', \cdot) \in R(j) \,:\, v' \in V'\})$$

If the local transition d is timed ($k = \mathbf{t}$), the global transition is timed (exponential) and we assign it rate parameter, r. Otherwise ($k = \mathbf{i}$), the global transition is immediate and we assign it the probability, $(r/W^j)/N$.

W^j is the sum of the weightings of all local enabled immediate transitions of agent j, $W^j := \sum_{(\cdot,\cdot,\cdot,r',\cdot,\cdot)\in D^l(a,\,j)} r'$, where:

$$D^l(a,\,j) = \{(x(j),\,\cdot,\,\mathbf{i},\,\cdot,\,S'_g,\,\cdot) \in D(a) \,:\, S'_g \subseteq R(j)\}$$

Thus r/W^j is the probability that the immediate transition corresponding to d fires, given that one of agent j's immediate transitions fires. N is the total number of agents with at least one local immediate transition enabled in this global state, $N := \sum_{i\in\mathcal{I}} \mathbf{1}_{W^i\neq 0}$. So choosing $(r/W^j)/N$ for the probability of the global immediate transition means that if two or more agents enable local immediate transitions simultaneously, the probability that one of a particular agent's local immediate transitions fires first is uniformly distributed among these agents.

C. Transaction start states

Before we may proceed we must define formally what it means for a global state to be a transaction start state. We do this by introducing the notion of a *transaction binding* in the next section. We must then also properly introduce *selection policies*, in the section that follows. Finally, in Section IV-C3, we give the outgoing global transitions from a transaction start state, completing the definition of the stochastic semantics of an STMC model.

A transaction start state is a global state in which one or more transaction specifications is *feasible*. Informally, a transaction specification is feasible if there are agents ready to begin the transaction, who together, offer all of the required roles.

1) Transaction bindings: More formally, let $(t,\,\mathcal{R}_t) \in T$ be a transaction specification and let $g = i \in \mathcal{I} \to (x(i),\,R(i),\,V(i))$ be a global state. Then $(t,\,\mathcal{R}_t)$ is feasible in g if and only if there exists at least one *transaction binding* for t, in g. A transaction binding, for t, in state g, is a function, say $b^g_t : \mathcal{R}_t \to \mathcal{I}$, which specifies an agent which can fulfil each of the required roles in $\mathcal{R}_t$. With each binding, we also associate a function $\hat{b}^g_t : \mathcal{R}_t \to N_A$ which specifies the type of the agent fulfilling each role, so $b^g_t(r) \in \mathcal{I}_{\hat{b}^g_t(r)}$, for each $r \in \mathcal{R}_t$.

Of course, for a transaction binding to make sense, each of the agents it specifies must actually be in a state where they offer the role in question. Recall from Section II that the transaction roles offered by agents of type $a \in N_A$ are specified by the set $T^{>}(a)$. Specifically, each element, $(q,\,f,\,o,\,y,\,B) \in T^{>}(a)$ specifies that when an agent of type a is in state q, it offers to play the role o in a transaction of type f. B specifies the variables to which other agents participating in the shared transaction will be bound, and y specifies the state into which the agent will move immediately if and when this shared transaction starts (the transactional immediate transition).

So if b^g_t is a transaction binding, it must be that for all $r \in \mathcal{R}_t$, there is an element, $(q,\,f,\,o,\,y,\,B) \in T^{>}(\hat{b}^g_t(r))$, such that, the agent is in the correct state, $q = x(b^g_t(r))$; it is offering the transaction in question, $f = t$ and it is offering to play the role in question, $o = r$. Note that this element of $T^{>}(\hat{b}^g_t(r))$ is unique if it exists (see Section II-A2), so that y and B are specified uniquely.

Furthermore, we also make the natural restriction on possible transaction bindings, that an individual agent cannot play more than one role in the same transaction, i.e. there do not exist distinct $r_1,\,r_2 \in \mathcal{R}_t$, such that $b^g_t(r_1) = b^g_t(r_2)$.

Consider again the example of Section III-A. Let $g \in \mathbf{G}$ be a global state in which exactly 10 client agents, say $c_1,\,\ldots,\,c_{10} \in \mathcal{I}_{\mathbf{SimpleClient}}$, and exactly 5 server agents, say $s_1,\,\ldots,\,s_5 \in \mathcal{I}_{\mathbf{SimpleServer}}$ are in their respective transaction entry points for the $\mathbf{Proc}$ transaction. This yields 50 distinct bindings, $\{b_{ij}\}$, for the $\mathbf{Proc}$ transaction in state g, where:

$$b_{ij}(client) = c_i$$
$$b_{ij}(server) = s_j$$

for each $1 \leq i \leq 10$ and $1 \leq j \leq 5$.

Out of all of the transaction bindings for feasible transactions, it is necessary to decide probabilistically which will be allowed to proceed first. This is the role of the *selection policy*, which was mentioned briefly earlier. Imposing this ordering is necessary since two or more feasible transactions may not all be able to begin in situations where there is contention for agents to play the required roles. In the context of the above example, this occurs when the number of clients awaiting processing outnumber the servers ready to accept requests.

2) Selection policies: In any transaction start state, $g \in \mathbf{G}$, there exists one or more pairs, $(t,\,b^g_t)$, where t is a transaction name and b^g_t is a binding for t, in state g. Write $B(g)$ for the set of all such pairs. $B(g)$ represents all of the shared transactions which are able to begin now. The role of a selection policy is to assign a probability to each pair representing the probability that this shared transaction begins next.

Formally, a selection policy is simply a function on the global state space, $\mathbf{G}$, which returns a probability distribution on $B(g)$. The simplest selection policy is the *totally uniform* selection policy, p^u. This simply treats every transaction and agent the same, assigning the uniform distribution to $B(g)$.

Implicit in the definition of selection policies is the requirement that they are *non-historical*. This means they consider only the current state of agents, and not, for example, their arrival order into transaction entry points.

3) Outgoing global transitions: Consider a state, $g = i \in \mathcal{I} \rightarrow (x(i), R(i), V(i)) \in \mathbf{G}$, in the global state space. Assume one or more transaction specifications are feasible, that is, g is a transaction start state. Assume we are using a selection policy, P. We now proceed to finally define the global transitions out of this state, as we have already done for simple states, thus completing the definition of the stochastic semantics of STMC models.

Recall that the set $B(g)$ represents the shared transactions ready to begin in state g. Since g is a transaction start state, $B(g)$ is non-empty. Each element of $B(g)$, contributes exactly one outgoing global immediate transition from state g, corresponding to that shared transaction starting. We wish to enable all of these transitions, weighted according to the selection policy, so that each shared transaction has the correct probability of starting next.

Specifically, consider an arbitrary element, $(t, b_t^g) \in B(g)$. We need to describe exactly the global immediate transition which fires if that shared transaction starts. In order to do this we must consider the local states the participating agents are in after it starts. Consider again the example of Section III-A. Immediately after a **Proc** transaction between a **SimpleClient** agent and a **SimpleServer** agent starts, the **SimpleClient** agent is in the *Send* state, and the **SimpleServer** agent is in the *Wait* state.

In general, the agents participating in the shared transaction, (t, b_t^g) are those specified by the binding:

$$\{b_t^g(r) \,:\, r \in \mathcal{R}_t\}$$

As discussed in Section IV-C1, for each $r \in \mathcal{R}_t$, there is a unique element, say, $(x_r, t_r, o_r, y_r, B_r) \in T^{>}(\hat{b}_t^g(r))$, where $x_r = x(b_t^g(r))$, $t_r = t$ and $o_r = r$. Recall that B_r specifies the variables that agent $b_t^g(r)$ will use to refer to the other agents taking part in this shared transaction, should it go ahead. y_r specifies the local state agent $b_t^g(r)$ will be in after beginning this shared transaction.

Thus the end state of the global transition corresponding to the shared transaction, (t, b_t^g) is $g_e = i \in \mathcal{I} \rightarrow (x_e(i), R_e(i), V_e(i))$, where g_e is the same as g except for:

- For each $r \in \mathcal{R}_t$, $x_e(b_t^g(r)) = y_r$ because agent $b_t^g(r)$ has moved to state y_r immediately after the shared transaction begins;
- We must update the variable bindings for each agent taking part in this newly-initiated shared transaction. So for each $r \in \mathcal{R}_t$, we must add to $V_e(b_t^g(r))$, the variable bindings for all of the other roles, $\mathcal{R}_t \setminus \{r\}$, played by other agents. For each $r' \in \mathcal{R}_t \setminus \{r\}$, the element $(v', r') \in B_r$ specifies that the variable v' will be used to refer to whichever other agent is playing the role of r'. Thus we add the element $(v', b_t^g(r'))$ to $V_e(b_t^g(r))$ in order to record this in the global state space.

As mentioned above, we set the probability of this global immediate transition to be the probability that the shared transaction it corresponds to, (t, b_t^g), is the next to start. This probability is determined by the selection policy as $P(g)[(t, b_t^g)]$.

V. Fluid analysis examples

A. Simple client/server model

We consider again the model introduced in Section III-A using the totally uniform selection policy. As before, the formal definition of the entire model specification is given in the appendix of the extended technical report [21].

Let T_c be the **Proc** transaction entry point in **SimpleClient** and T_s, the **Proc** transaction entry point in **SimpleServer**. Then we consider a sequence of model configurations, $\{\mathbf{C}_i\}_{i=1}^{\infty}$, where:

$$\mathbf{C}_i(\mathbf{SimpleClient}, T_c) = 2i$$
$$\mathbf{C}_i(\mathbf{SimpleServer}, T_s) = i$$

and $\mathbf{C}_i$ is zero everywhere else. That is, this sequence of configurations represents the sequence of models with increasing client and server populations, but where there are twice as many clients as there are servers. All agents begin in their respective transaction entry points.

In the following table, we define stochastic processes which count the numbers of agents in specific states. Formally, each should be indexed by i, corresponding to the particular process for the model defined by the earlier specification and the configuration, $\mathbf{C}_i$. However, we suppress this index to reduce the necessary notation.

$\mathbf{C}_{\mathbf{Proc}}(t)$	Counts the number of **SimpleClient** agents either in the transaction entry point or the state *Send*
$\mathbf{C}_{Task}(t)$	Counts the number of **SimpleClient** agents in the state *Task*
$\mathbf{S}_{\mathbf{Proc}}(t)$	Counts the number of **SimpleServer** agents either in the transaction entry point or the state *Wait*
$\mathbf{S}_{Reset}(t)$	Counts the number of **SimpleServer** agents in the state *Reset*
$\mathbf{T}_{\mathbf{sent}}(t)$	Counts the number of **SimpleServer** and **SimpleClient** agent transaction pairs where the **SimpleServer** is in the *Process* state and the **SimpleClient** is in the *Wait* state

Then a lumpability argument can be used to show that the joint stochastic process:

$$(\mathbf{C}_{\mathbf{Proc}}(t), \mathbf{C}_{Task}(t), \mathbf{S}_{\mathbf{Proc}}(t), \mathbf{S}_{Reset}(t), \mathbf{T}_{\mathbf{sent}}(t))$$

is a CTMC. The possible transitions from each state, with their rates, are:

$$(\mathbf{C}_{\mathbf{Proc}}, \mathbf{C}_{Task}, \mathbf{S}_{\mathbf{Proc}}, \mathbf{S}_{Reset}, \mathbf{T}_{\mathbf{sent}}) \xrightarrow{\min(\mathbf{C}_{\mathbf{Proc}}, \mathbf{S}_{\mathbf{Proc}})\lambda} (\mathbf{C}_{\mathbf{Proc}} - 1, \mathbf{C}_{Task}, \mathbf{S}_{\mathbf{Proc}} - 1, \mathbf{S}_{Reset}, \mathbf{T}_{\mathbf{sent}} + 1)$$

$$(\mathbf{C}_{\mathbf{Proc}}, \mathbf{C}_{Task}, \mathbf{S}_{\mathbf{Proc}}, \mathbf{S}_{Reset}, \mathbf{T}_{\mathbf{sent}}) \xrightarrow{\mathbf{T}_{\mathbf{sent}}\gamma} (\mathbf{C}_{\mathbf{Proc}}, \mathbf{C}_{Task} + 1, \mathbf{S}_{\mathbf{Proc}}, \mathbf{S}_{Reset} + 1, \mathbf{T}_{\mathbf{sent}} - 1)$$

$$(\mathbf{C}_{\mathbf{Proc}}, \mathbf{C}_{Task}, \mathbf{S}_{\mathbf{Proc}}, \mathbf{S}_{Reset}, \mathbf{T}_{\mathbf{sent}}) \xrightarrow{\mathbf{C}_{Task}\mu} (\mathbf{C}_{\mathbf{Proc}} + 1, \mathbf{C}_{Task} - 1, \mathbf{S}_{\mathbf{Proc}}, \mathbf{S}_{Reset}, \mathbf{T}_{\mathbf{sent}})$$

$$(\mathbf{C}_{\mathbf{Proc}}, \mathbf{C}_{Task}, \mathbf{S}_{\mathbf{Proc}}, \mathbf{S}_{Reset}, \mathbf{T}_{\mathbf{sent}}) \xrightarrow{\mathbf{S}_{Reset}\rho} (\mathbf{C}_{\mathbf{Proc}}, \mathbf{C}_{Task}, \mathbf{S}_{\mathbf{Proc}} + 1, \mathbf{S}_{Reset} - 1, \mathbf{T}_{\mathbf{sent}})$$

We now consider, for each component of this joint stochastic process, the rate at which it is incremented and the rate at which it is decremented. Subtracting the second quantity from the first yields an approximate 'rate of change' for this quantity and suggests the following system of differential equations to define a fluid approximation to the state of the system at time t:

$$\dot{C}_{\mathbf{Proc}}(t) = -\min(C_{\mathbf{Proc}}(t),\, S_{\mathbf{Proc}}(t))\lambda + C_{Task}(t)\mu$$
$$\dot{C}_{Task}(t) = -C_{Task}(t)\mu + T_{\mathbf{sent}}(t)\gamma$$
$$\dot{S}_{\mathbf{Proc}}(t) = -\min(C_{\mathbf{Proc}}(t),\, S_{\mathbf{Proc}}(t))\lambda + S_{Reset}(t)\rho$$
$$\dot{S}_{Reset}(t) = -S_{Reset}(t)\rho + T_{\mathbf{sent}}(t)\gamma$$
$$\dot{T}_{\mathbf{sent}}(t) = -T_{\mathbf{sent}}(t)\gamma + \min(C_{\mathbf{Proc}}(t),\, S_{\mathbf{Proc}}(t))\lambda$$

Figure 2 shows various model quantities, computed using this fluid approximation (with initial conditions to match the corresponding model configuration). This is compared with the expectation computed through repeated stochastic simulation of the CTMC. In the figure legend, "Clients blocked" is the number of **SimpleClient** agents blocked waiting in the transaction entry point for a server. In terms of the above model quantities this is:

$$\mathbf{C}_{\mathbf{Proc}}(t) - \min(\mathbf{C}_{\mathbf{Proc}}(t),\, \mathbf{S}_{\mathbf{Proc}}(t))$$

"Servers busy" is the number of **SimpleServer** agents in either the *Wait* or *Process* state:

$$\min(\mathbf{C}_{\mathbf{Proc}}(t),\, \mathbf{S}_{\mathbf{Proc}}(t)) + \mathbf{T}_{\mathbf{sent}}(t)$$

Finally, "Servers resetting" is the number of **SimpleServer** agents in the *Reset* state, that is, $\mathbf{S}_{Reset}(t)$.

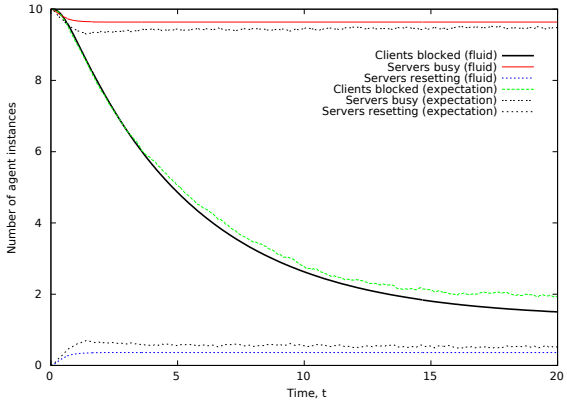
We see that the fluid approximations are a close approximation to the actual expectations in both cases. The fact that the approximation improves in terms of the error relative to the population size as the population size increases is to be expected by the results of Kurtz [22] and others [23].

B. Client/server model with two-phase service

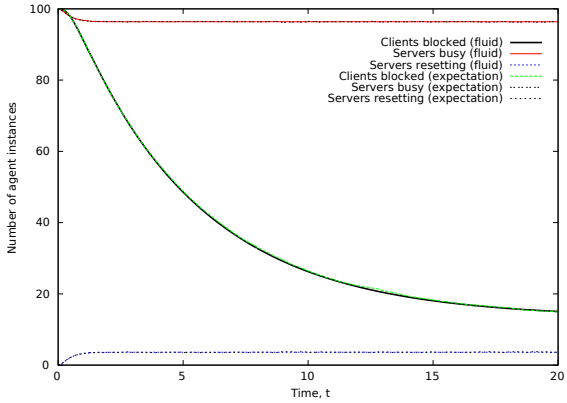
This is a straightforward modification of the previous model where we replace **SimpleServer** with a new agent type, **SimpleServer2P**. **SimpleServer2P** models a server that processes the client's request as a two-phase operation. The interface between the two agents is unchanged, which is reflected in the fact that the model uses the same transaction specifications, signal names and variables as in the first example. The graphical representation of **SimpleServer2P** is given in Figure 3 and we use the same agent type to represent the client, **SimpleClient**, as above (Figure 1). Again, we use the totally uniform selection policy. The formal definition of the entire model specification is again given in the appendix of the extended technical report [21].

Similarly to the previous example, let T_c be the transaction entry point in **SimpleClient** and T_s, the transaction entry point in **SimpleServer2P**. Then, as before, we consider a sequence of model configurations $\{\mathbf{C}_i\}_{i=1}^{\infty}$, where:

$$\mathbf{C}_i(\mathbf{SimpleClient},\, T_c) = 2i$$
$$\mathbf{C}_i(\mathbf{SimpleServer2P},\, T_s) = i$$



(a) Configuration: $\mathbf{C}_{10}$.



(b) Configuration: $\mathbf{C}_{100}$.

Fig. 2: Fluid approximation and expectation comparison for simple client/server model. Rates: $\lambda = 3.0$, $\gamma = 0.2$, $\rho = 5.0$, $\mu = 0.2$.

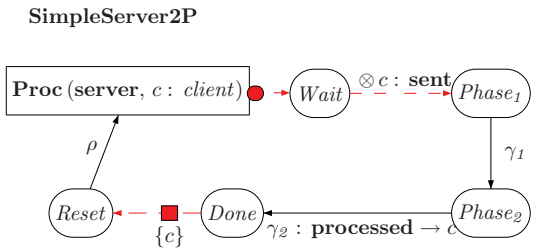
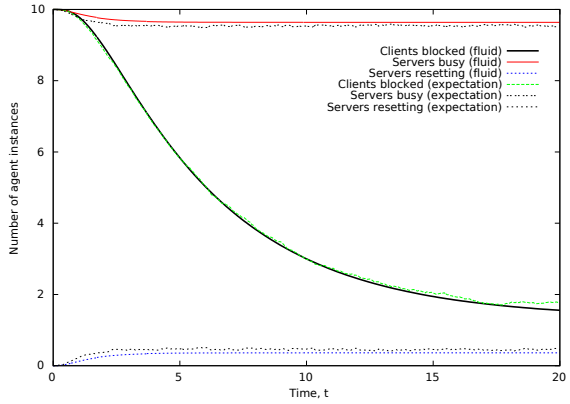


Fig. 3: Agent type for two-phase server.

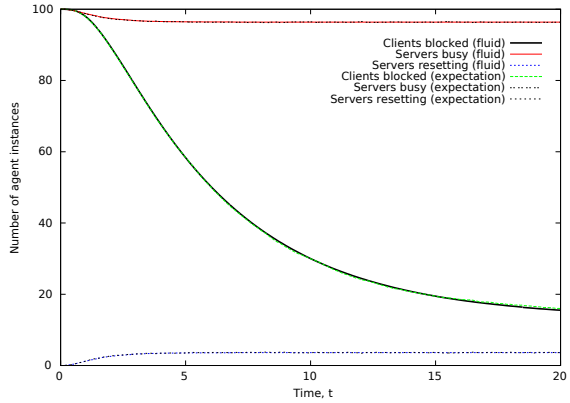
and $\mathbf{C}_i$ is zero everywhere else.

We define the analogous component counting stochastic processes as in the previous example, except for $\mathbf{T}_{\mathbf{sent}}(t)$, which we replace with $\mathbf{T}_{Phase_1}(t)$ and $\mathbf{T}_{Phase_2}(t)$ to count the number of **SimpleServer2P** and **SimpleClient** agent transaction pairs where the **SimpleServer2P** is in the $Phase_1$, respectively, $Phase_2$ state and the **SimpleClient** is in the *Wait* state.

Applying a lumpability argument, we obtain the following



(a) Configuration: $\mathbf{C}_{10}$.



(b) Configuration: $\mathbf{C}_{100}$.

Fig. 4: Fluid approximation and expectation comparison for simple client/server model with two-phase service. Rates: $\lambda = 3.0$, $\gamma_1 = 0.4$, $\gamma_2 = 0.4$, $\rho = 5.0$, $\mu = 0.2$.

system of differential equations as a fluid approximation:

$$\begin{aligned}
\dot{C}_{\mathbf{Proc}}(t) &= -\min(C_{\mathbf{Proc}}(t),\, S_{\mathbf{Proc}}(t))\lambda + C_{Task}(t)\mu \\
\dot{C}_{Task}(t) &= -C_{Task}(t)\mu + T_{Phase_2}(t)\gamma_2 \\
\dot{S}_{\mathbf{Proc}}(t) &= -\min(C_{\mathbf{Proc}}(t),\, S_{\mathbf{Proc}}(t))\lambda + S_{Reset}(t)\rho \\
\dot{S}_{Reset}(t) &= -S_{Reset}(t)\rho + T_{Phase_2}(t)\gamma_2 \\
\dot{T}_{Phase_1}(t) &= -T_{Phase_1}(t)\gamma_1 + \min(C_{\mathbf{Proc}}(t),\, S_{\mathbf{Proc}}(t))\lambda \\
\dot{T}_{Phase_2}(t) &= -T_{Phase_2}(t)\gamma_2 + T_{Phase_1}(t)\gamma_1
\end{aligned}$$

Figure 4 shows a comparison between the analogous model quantities as in Figure 2 for the first example, both for fluid approximations and the actual expectations. We have deliberately chosen $\gamma_1 = \gamma_2 = 0.4 = 2\gamma$, so that the mean processing time is the same as in the single-phase example. The two-phase nature of the processing time distribution in the second example serves then to reduce the variance of the processing time. We note that in Figure 4a, the correspondence between the fluid approximation and the actual expectation is better than that of Figure 2a, even though they both correspond to the same size of client and server populations. The reduced variance of the processing time is one plausible explanation for this.

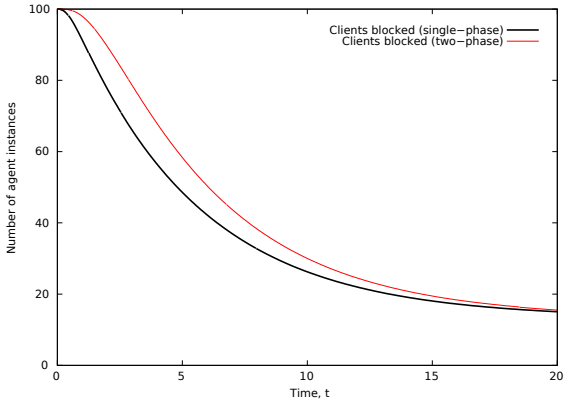


Fig. 5: Fluid limit of number of blocked clients comparison for single-phase and two-phase processing time distributions.

Figure 5 shows a comparison of the fluid approximation for the blocked clients from Figure 2b and Figure 4b on the same graph. This illustrates more clearly that, even though the processing time has the same mean duration in each case, the difference in variance does effect the model, even in the fluid limit. This is in no way paradoxical; indeed in the fluid limit, the relative variability of the *counting processes* does approach zero, however the variability of the state of *individual agents* of course does not and may effect the deterministic limit of the counting processes.

C. Client/server model with two server types

For this example, we consider a model where there are agents of all three types, **SimpleClient**, **SimpleServer** and **SimpleServer2P** (Figures 1 and 3) all together in the same model. This represents the scenario of clients who can be served by two different types of server. One server proceeds with one-phase service and the other with two-phase service. The formal definition of the entire model specification is again given in the appendix of the extended technical report [21].

Using the totally uniform selection policy, the probability of a client being served by either type of server will depend only on how many of each type are ready to offer service. However, we note that a priority-based selection policy differentiating between potential transactions might be appropriate to capture certain scenarios.

The purpose of this example is to illustrate a case where the differential equation fluid limit construction of the previous two examples cannot be readily applied. Therefore we define a direction for future work in developing alternative fluid limit constructions, where possible. The following discussion applies for any model configuration so we do not formally define one here.

One cannot aggregate states together in the same manner as in the previous two examples. This is because information concerning the interleaving order of the two classes of servers is lost. The reason for this is that we count being in the transaction entry point, and the first state of the transaction (after

the transactional immediate transitions have fired), together. This is necessary to obtain a CTMC without vanishing states, upon which the fluid analysis of the previous two examples can be performed.

Of course, if the state of waiting for a transaction and being in the first state of a transaction is counted separately, then a lumped CTMC with vanishing states is obtained. Fluid analysis as in the previous examples cannot be applied to such a CTMC since the rates between states would be discontinuous functions of the counting variables. However, this CTMC can still be analysed using existing techniques, where the size of the discrete state space does not make this intractable.

D. A hierarchy of servers

Finally, we present a slightly more complicated example, for which we will be able to construct differential equation-based fluid limits. The idea behind this example is that again, clients must have requests processed by servers, after which they can perform some task with the result independently. The request can be processed by one of two types of server. Initially, the client tries to begin a transaction with one of a class of primary servers. If, after some random timeout, it has not managed to begin such a transaction, it will give up and fall-back to trying to begin a transaction with one of a class of secondary servers. We will model the secondary servers as more plentiful, but with a longer expected duration to process the request (in a real world scenario, one might imagine they are cheaper).

We distinguish the idea of primary and secondary service by using two transaction specifications:

$$(\mathbf{Proc_p}, \{client, server_p\})$$
$$(\mathbf{Proc_s}, \{client, server_s\})$$

The $\mathbf{Proc_p}$ transaction requires one agent to play the client role ($client$) and one to play the role of the primary server ($server_p$). The $\mathbf{Proc_s}$ transaction again requires one agent to play the client role ($client$) and another to play the role of the secondary server ($server_s$). We use the same signal and variable names as in all of the previous examples. Again, we use the totally uniform selection policy.

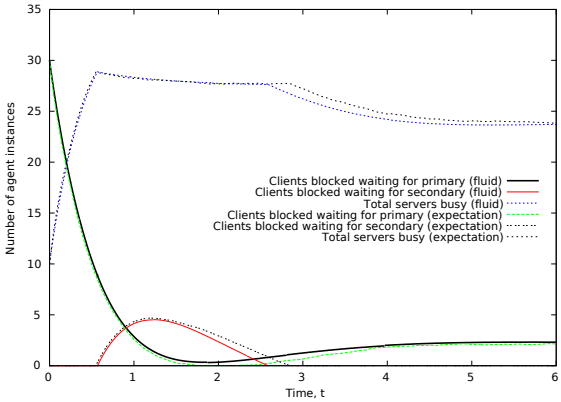
We will model primary servers as agents of the new type, $\mathbf{SimpleServer_p}$, which is the same as $\mathbf{SimpleServer}$ in previous examples, except for it offering the $\mathbf{Proc_p}$ transaction. So our primary servers complete the request service with one phase. Secondary servers are modelled as agents of the new type, $\mathbf{SimpleServer_s}$, which is the same as $\mathbf{SimpleServer2P}$ in previous examples, again, except for it offering the $\mathbf{Proc_s}$ transaction. So our secondary servers take two phases to complete the service. We must introduce a new client agent type, $\mathbf{ClientTimeout}$, to model the timeout behaviour. All three of these agent types are shown in Figure 7. As in the previous examples the formal definition of the entire model specification is given in the appendix of the extended technical report [21].

Like in previous examples, let T_{cp} be the transaction entry point in $\mathbf{ClientTimeout}$ where the client is attempting to begin a transaction with a $\mathbf{SimpleServer_p}$, and T_{cs} be the transaction entry point in $\mathbf{ClientTimeout}$ where the client is attempting to begin a transaction with a $\mathbf{SimpleServer_s}$. Let T_{sp} be the transaction entry point in $\mathbf{SimpleServer_p}$ and T_{ss} the transaction entry point in $\mathbf{SimpleServer_s}$. Then, similarly to the previous example, we consider a sequence of model configurations $\{\mathbf{C}_i\}_{i=1}^{\infty}$, where:

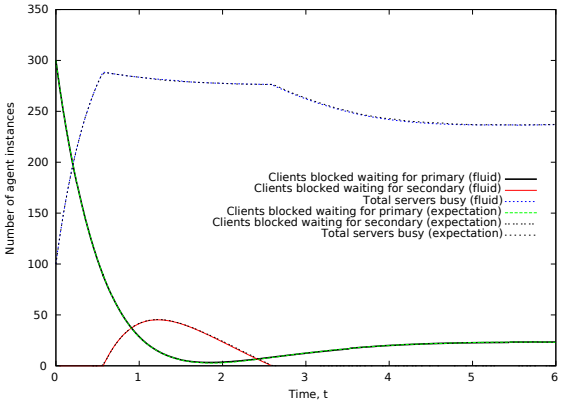
$$\mathbf{C}_i(\mathbf{ClientTimeout}, T_{cp}) = 4i$$
$$\mathbf{C}_i(\mathbf{SimpleServer_p}, T_{sp}) = i$$
$$\mathbf{C}_i(\mathbf{SimpleServer_s}, T_{ss}) = 2i$$

and $\mathbf{C}_i$ is zero everywhere else. This represents a third more clients than total servers, with two thirds of the servers of the slower, $\mathbf{SimpleServer_s}$ type, and the other third, the faster $\mathbf{SimpleServer_p}$ type.

We now define, in the following table, analogous stochastic counting processes as in the first two examples.



(a) Configuration: $\mathcal{C}_{10}$**.**



(b) Configuration: $\mathcal{C}_{100}$**.**

Fig. 6: Fluid approximation and expectation comparison for hierarchical client/server model. Rates: $\lambda = 3.0$**,** $\gamma = 1.0$**,** $\gamma_1 = 2/3$**,** $\gamma_2 = 2/3$**,** $\rho = 5.0$**,** $\mu = 0.8$**,** $\nu = 2.0$**.**

$\mathbf{C}_{\mathbf{Proc_p}}(t)$	Counts the number of **ClientTimeout** agents either in the transaction entry point for $\mathbf{Proc_p}$ or the state *Send*, after having begun this transaction
$\mathbf{C}_{\mathbf{Proc_s}}(t)$	Counts the number of **ClientTimeout** agents either in the transaction entry point for $\mathbf{Proc_s}$ or the state *Send*, after having begun this transaction
$\mathbf{S}^p_{\mathbf{Proc_p}}(t)$	Counts the number of $\mathbf{SimpleServer_p}$ agents either in their transaction entry point or the state *Wait*
$\mathbf{S}^s_{\mathbf{Proc_s}}(t)$	Counts the number of $\mathbf{SimpleServer_s}$ agents either in their transaction entry point or the state *Wait*
$\mathbf{C}_{Task}(t)$	Counts the number of **ClientTimeout** agents in the state *Task*
$\mathbf{S}^p_{Reset}(t)$	Counts the number of $\mathbf{SimpleServer_p}$ agents in the state *Reset*
$\mathbf{S}^s_{Reset}(t)$	Counts the number of $\mathbf{SimpleServer_s}$ agents in the state *Reset*
$\mathbf{T}^p_{\mathbf{sent}}(t)$	Counts the number of $\mathbf{SimpleServer_p}$ and **ClientTimeout** agent transaction pairs where the $\mathbf{SimpleServer_p}$ is in the *Process* state and the **ClientTimeout** is in the *Wait* state
$\mathbf{T}^s_{Phase_1}(t)$	Counts the number of $\mathbf{SimpleServer_s}$ and **ClientTimeout** agent transaction pairs where the $\mathbf{SimpleServer_s}$ is in the $Phase_1$ state and the **ClientTimeout** is in the *Wait* state
$\mathbf{T}^s_{Phase_2}(t)$	Counts the number of $\mathbf{SimpleServer_s}$ and **ClientTimeout** agent transaction pairs where the $\mathbf{SimpleServer_s}$ is in the $Phase_2$ state and the **ClientTimeout** is in the *Wait* state

Applying again a lumpability argument, we obtain the following system of differential equations as a fluid approximation:

$$\begin{aligned}
\dot{C}_{\mathbf{Proc_p}}(t) = & -\min(C_{\mathbf{Proc_p}}(t),\, S^p_{\mathbf{Proc_p}}(t))\lambda - T(t)\nu \\
& + C_{Task}(t)\mu \\
\dot{C}_{\mathbf{Proc_s}}(t) = & -\min(C_{\mathbf{Proc_s}}(t),\, S^s_{\mathbf{Proc_s}}(t))\lambda + T(t)\nu \\
\dot{S}^p_{\mathbf{Proc_p}}(t) = & -\min(C_{\mathbf{Proc_p}}(t),\, S^p_{\mathbf{Proc_p}}(t))\lambda \\
& + S^p_{Reset}(t)\rho \\
\dot{S}^s_{\mathbf{Proc_s}}(t) = & -\min(C_{\mathbf{Proc_s}}(t),\, S^s_{\mathbf{Proc_s}}(t))\lambda \\
& + S^s_{Reset}(t)\rho \\
\dot{C}_{Task}(t) = & -C_{Task}(t)\mu + T^p_{\mathbf{sent}}(t)\gamma \\
& + T^s_{Phase_2}(t)\gamma_2 \\
\dot{S}^p_{Reset}(t) = & -S^p_{Reset}(t)\rho + T^p_{\mathbf{sent}}(t)\gamma \\
\dot{S}^s_{Reset}(t) = & -S^s_{Reset}(t)\rho + T^s_{Phase_2}(t)\gamma_2 \\
\dot{T}^p_{\mathbf{sent}}(t) = & -T^p_{\mathbf{sent}}(t)\gamma \\
& + \min(C_{\mathbf{Proc_p}}(t),\, S^p_{\mathbf{Proc_p}}(t))\lambda \\
\dot{T}^s_{Phase_1}(t) = & -T^s_{Phase_1}(t)\gamma_1 \\
& + \min(C_{\mathbf{Proc_s}}(t),\, S^s_{\mathbf{Proc_s}}(t))\lambda \\
\dot{T}^s_{Phase_2}(t) = & -T^s_{Phase_2}(t)\gamma_2 + T^s_{Phase_1}(t)\gamma_1
\end{aligned}$$

where $T(t)$ counts the number of **ClientTimeout** agents blocked waiting for a $\mathbf{SimpleServer_p}$:

$$T(t) := C_{\mathbf{Proc_p}}(t) - \min(C_{\mathbf{Proc_p}}(t),\, S^p_{\mathbf{Proc_p}}(t))$$

Figure 6 shows a comparison between similar model quantities as in Figures 2 and 4, both for fluid approximations and the actual expectations. Specifically, in the figure legend, "Clients blocked waiting for primary" is the number of **ClientTimeout** agents blocked waiting in the transaction entry point for a $\mathbf{SimpleServer_p}$ (that is, $T(t)$, defined above). "Clients blocked waiting for secondary" is the number of **ClientTimeout** agents blocked waiting in the transaction entry point for a $\mathbf{SimpleServer_s}$:

$$C_{\mathbf{Proc_s}}(t) - \min(C_{\mathbf{Proc_s}}(t),\, S^s_{\mathbf{Proc_s}}(t))$$

"Total servers busy" is simply the combined number of both **SimpleServer** and **SimpleServer2P** agents engaged in a transaction with a client agent:

$$\begin{aligned}
\min(C_{\mathbf{Proc_p}}(t),\, S^p_{\mathbf{Proc_p}}(t)) + \min(C_{\mathbf{Proc_s}}(t),\, S^s_{\mathbf{Proc_s}}(t)) \\
+ T^p_{\mathbf{sent}}(t) + T^s_{Phase_1}(t) + T^s_{Phase_2}(t)
\end{aligned}$$

VI. Conclusion

In this paper, we have introduced a new low level performance formalism, STMCs, which captures massively parallel systems and multi-phase component cooperation. We have shown through several examples how fluid performance analysis can be achieved for very large systems[3] where contention for resources is accurately represented, not by a race condition as is often the choice in Markovian formalisms, but by a probabilistic selection. We have discussed how different selection policies on resources could be expressed for future models.

We have also shown how adopting the probabilistic selection approach to synchronisation can lead to models where the fluid approximation by differential equation route is not immediately applicable, defining a potential direction for future research. Specifically, such models include those where transactions may be entered into with different agent types offering the same role simultaneously. For example, when different classes of servers are able to process a request. We intend to characterise the classes of STMC models which do admit fluid analysis, both in the differential equation sense discussed in this paper, and potentially also in others. In particular, we are interested in developing functional central limit theorems for STMCs (for example, systems of stochastic differential equations) which may provide a better approximation than ordinary differential equations.

We have not yet properly addressed issues of model correctness or well-formedness here as we wished to present a broader picture of the formalism to start with. These issues will be addressed once the feature list for the formalism is stable. We have presented a set of performance modelling features that we believe should be easily accessible in the next generation of performance modelling formalisms given recent exciting developments in fluid analysis techniques.

STMCs are not designed to be a competitor behavioural formalism to the SPAs and SPNs that have been developed

[3]Of the order of many billions of discrete states.

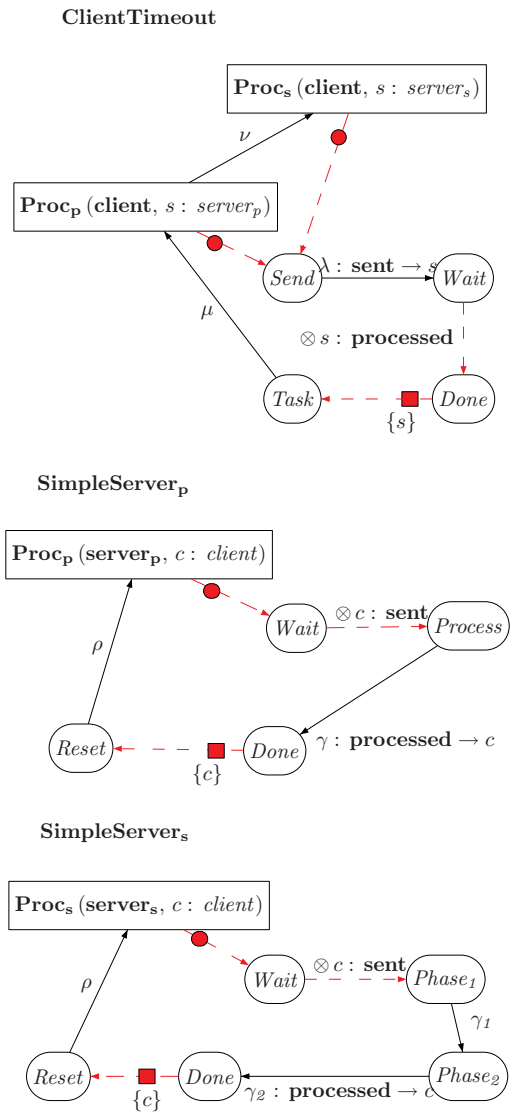


Fig. 7: Agent types for client and servers in hierarchical client/server example.

over many years. Indeed we expect to be able to generate translations between these formalisms and the abstraction provided by STMCs.

REFERENCES

[1] M. Ajmone Marsan, G. Conte, and G. Balbo, "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems," *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 93–122, May 1984.

[2] J. Hillston, *A Compositional Approach to Performance Modelling*, ser. Distinguished Dissertations in Computer Science. Cambridge University Press, 1996, vol. 12.

[3] M. Bernardo and R. Gorrieri, "Extended Markovian Process Algebra," in *CONCUR'96, Proceedings of the 7th International Conference on Concurrency Theory*, ser. Lecture Notes in Computer Science, U. Montanari and V. Sassone, Eds., vol. 1119. Springer-Verlag, Pisa, August 1996, pp. 315–330.

[4] H. Hermanns, "Interactive Markov chains," Ph.D. dissertation, Universität Erlangen-Nürnberg, July 1998.

[5] B. Plateau and J.-M. Fourneau, "A methodology for solving Markov models of parallel systems," *Journal of Parallel and Distributed Computing*, vol. 12, no. 4, pp. 370–387, August 1991.

[6] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior, and application," in *International Workshop on Timed Petri Nets*. IEEE, 1985, pp. 106–115.

[7] W. J. Knottenbelt and P. G. Harrison, "Distributed disk-based solution techniques for large Markov models," in *NSMC'99, Proceedings of the 3rd Intl. Conference on the Numerical Solution of Markov Chains*, Zaragoza, September 1999, pp. 58–75.

[8] N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt, "Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, pp. 908–920, August 2004.

[9] ——, "Response time densities in Generalised Stochastic Petri Net models," in *WOSP 2002, Proceedings of the 3rd International Workshop on Software and Performance*, Rome, July 2002, pp. 46–54.

[10] J. Hillston, "Fluid flow approximation of PEPA models," in *QEST'05, Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*. Torino: IEEE Computer Society Press, September 2005, pp. 33–42.

[11] L. Bortolussi and A. Policriti, "Stochastic concurrent constraint programming and differential equations," in *QAPL'07, 5th Workshop on Quantitative Aspects of Programming Languages*, ser. Electronic Notes in Theoretical Computer Science, vol. 190(3), September 2007, pp. 27–42.

[12] R. A. Hayden and J. T. Bradley, "Fluid semantics for passive stochastic process algebra cooperation," in *VALUETOOLS'08, Third International Conference on Performance Evaluation Methodologies and Tools*, Athens, 2008.

[13] J. Julvez, E. Jimenez, L. Recalde, and M. Silva, "On observability in timed continuous Petri net systems," in *QEST'04, Proceedings of the 1st International Conference on Quantitative Evaluation of Systems*. Twente: IEEE, September 2004, pp. 60–69.

[14] J. E. Neilson, C. M. Woodside, D. C. Petriu, and S. Majumdar, "Software bottlenecking in client-server systems and rendezvous networks," *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 776–782, September 1995.

[15] G. Franks and C. M. Woodside, "Performance of multi-level client-server systems with parallel service operations," in *WOSP 1998, Proceeding of 1st International Workshop on Software Performance*. Sante Fe, New Mexico: ACM, October 1998, pp. 120–130.

[16] J. T. Bradley, N. J. Dingle, W. J. Knottenbelt, and P. G. Harrison, "Performance queries on semi-Markov stochastic Petri nets with an extended Continuous Stochastic Logic," in *PNPM'03, Proceedings of Petri Nets and Performance Models*, G. Ciardo and W. Sanders, Eds. University of Illinois at Urbana-Champaign: IEEE Computer Society, September 2003, pp. 62–71.

[17] J. T. Bradley, "Semi-Markov PEPA: Compositional modelling and analysis with generally distributed actions," in *UKPEW'04, Proceedings of 20th Annual UK Performance Engineering Workshop*, I. Awan, Ed., University of Bradford, July 2004, pp. 266–275.

[18] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[19] C. Priami, "A stochastic π-calculus," in *Process Algebra and Performance Modelling Workshop*, ser. Special Issue: The Computer Journal, S. Gilmore and J. Hillston, Eds., vol. 38(7). CEPIS, Edinburgh, June 1995, pp. 578–589.

[20] L. Bortolussi, "Stochastic concurrent constraint programming," in *QAPL'06, 4th Workshop on Quantitative Aspects of Programming Languages*, ser. Electronic Notes in Theoretical Computer Science, vol. 164, 2006, pp. 65–80.

[21] R. Hayden and J. T. Bradley, "Shared transaction Markov chains for fluid analysis of massively parallel systems (extended version)," Imperial College London, Tech. Rep., 2009. [Online]. Available: http://pubs.doc.ic.ac.uk/stmc-extended

[22] T. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *Applied Probability*, vol. 7, no. 1, pp. 49–58, April 1970.

[23] R. W. R. Darling and J. R. Norris, "Differential equation approximations for Markov chains," *Probability Surveys*, vol. 5, pp. 37–79, 2008.

APPENDIX

For each of the four examples considered in this paper, we now give the exact definitions of the components making up the STMC specification 8-tuple:

$$(N_A,\ A,\ N_T,\ T,\ R,\ P,\ S,\ V)$$

A. Simple client/server model

$$\begin{aligned}
N_A &= \{\mathbf{SimpleClient},\ \mathbf{SimpleServer}\}\\
A &= \{(\mathbf{SimpleClient},\ X_1,\ D_1,\ T_1^{>},\ T_1^{<}),\\
&\quad\ (\mathbf{SimpleServer},\ X_2,\ D_2,\ T_2^{>},\ T_2^{<})\}\\
N_T &= \{\mathbf{Proc}\}\\
T &= \{(\mathbf{Proc},\ \{\mathit{client},\ \mathit{server}\})\}\\
R &= \{\mathit{client},\ \mathit{server}\}\\
P &= p^u\\
S &= \{\mathbf{sent},\ \mathbf{processed}\}\\
V &= \{c,\ s\}
\end{aligned}$$

where:

$$\begin{aligned}
X_1 &= \{T_c,\ \mathit{Send},\ \mathit{Wait},\ \mathit{Done},\ \mathit{Task}\}\\
D_1 &= \{(\mathit{Send},\ \mathit{Wait},\ \mathbf{t},\ \lambda,\ \emptyset,\ \{(s,\ \mathbf{sent})\}),\\
&\quad\ (\mathit{Wait},\ \mathit{Done},\ \mathbf{i},\ 1,\ \{(s,\ \mathbf{processed})\},\ \emptyset),\\
&\quad\ (\mathit{Done},\ \mathit{Task},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\\
&\quad\ (\mathit{Task},\ T_c,\ \mathbf{t},\ \mu,\ \emptyset,\ \emptyset)\}\\
T_1^{>} &= \{(T_c,\ \mathbf{Proc},\ \mathbf{client},\ \mathit{Send},\ \{(s,\ \mathit{server})\})\}\\
T_1^{<} &= \{((\mathit{Done},\ \mathit{Task},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\ s)\}
\end{aligned}$$

and:

$$\begin{aligned}
X_2 &= \{T_s,\ \mathit{Wait},\ \mathit{Process},\ \mathit{Done},\ \mathit{Reset}\}\\
D_2 &= \{(\mathit{Wait},\ \mathit{Process},\ \mathbf{i},\ 1,\ \{(c,\ \mathbf{sent})\},\ \emptyset),\\
&\quad\ (\mathit{Process},\ \mathit{Done},\ \mathbf{t},\ \gamma,\ \emptyset,\ \{(c,\ \mathbf{processed})\}),\\
&\quad\ (\mathit{Done},\ \mathit{Reset},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\\
&\quad\ (\mathit{Reset},\ T_s,\ \mathbf{t},\ \rho,\ \emptyset,\ \emptyset)\}\\
T_2^{>} &= \{(T_s,\ \mathbf{Proc},\ \mathbf{server},\ \mathit{Wait},\ \{(c,\ \mathit{client})\})\}\\
T_2^{<} &= \{((\mathit{Done},\ \mathit{Reset},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\ c)\}
\end{aligned}$$

B. Client/server model with two-phase service

$$\begin{aligned}
N_A &= \{\mathbf{SimpleClient},\ \mathbf{SimpleServer2P}\}\\
A &= \{(\mathbf{SimpleClient},\ X_1,\ D_1,\ T_1^{>},\ T_1^{<}),\\
&\quad\ (\mathbf{SimpleServer2P},\ X_3,\ D_3,\ T_3^{>},\ T_3^{<})\}\\
N_T &= \{\mathbf{Proc}\}\\
T &= \{(\mathbf{Proc},\ \{\mathit{client},\ \mathit{server}\})\}\\
R &= \{\mathit{client},\ \mathit{server}\}\\
P &= p^u\\
S &= \{\mathbf{sent},\ \mathbf{processed}\}\\
V &= \{c,\ s\}
\end{aligned}$$

where X_1, D_1, $T_1^{>}$ and $T_1^{<}$ are as above and:

$$\begin{aligned}
X_3 &= \{T_s,\ \mathit{Wait},\ \mathit{Phase}_1,\ \mathit{Phase}_2,\ \mathit{Done},\ \mathit{Reset}\}\\
D_3 &= \{(\mathit{Wait},\ \mathit{Phase}_1,\ \mathbf{i},\ 1,\ \{(c,\ \mathbf{sent})\},\ \emptyset),\\
&\quad\ (\mathit{Phase}_1,\ \mathit{Phase}_2,\ \mathbf{t},\ \gamma_1,\ \emptyset,\ \emptyset),\\
&\quad\ (\mathit{Phase}_2,\ \mathit{Done},\ \mathbf{t},\ \gamma_2,\ \emptyset,\ \{(c,\ \mathbf{processed})\}),\\
&\quad\ (\mathit{Done},\ \mathit{Reset},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\\
&\quad\ (\mathit{Reset},\ T_s,\ \mathbf{t},\ \rho,\ \emptyset,\ \emptyset)\}\\
T_3^{>} &= \{(T_s,\ \mathbf{Proc},\ \mathbf{server},\ \mathit{Wait},\ \{(c,\ \mathit{client})\})\}\\
T_3^{<} &= \{((\mathit{Done},\ \mathit{Reset},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\ c)\}
\end{aligned}$$

C. Client/server model with two server types

$$\begin{aligned}
N_A &= \{\mathbf{SimpleClient},\ \mathbf{SimpleServer},\ \mathbf{SimpleServer2P}\}\\
A &= \{(\mathbf{SimpleClient},\ X_1,\ D_1,\ T_1^{>},\ T_1^{<}),\\
&\quad\ (\mathbf{SimpleServer},\ X_2,\ D_2,\ T_2^{>},\ T_2^{<}),\\
&\quad\ (\mathbf{SimpleServer2P},\ X_3,\ D_3,\ T_3^{>},\ T_3^{<})\}\\
N_T &= \{\mathbf{Proc}\}\\
T &= \{(\mathbf{Proc},\ \{\mathit{client},\ \mathit{server}\})\}\\
R &= \{\mathit{client},\ \mathit{server}\}\\
P &= p^u\\
S &= \{\mathbf{sent},\ \mathbf{processed}\}\\
V &= \{c,\ s\}
\end{aligned}$$

where all of X_i, D_i, $T_i^{>}$ and $T_i^{<}$ for $i \in \{1,\ 2,\ 3\}$ are as defined earlier.

D. A hierarchy of servers

$$\begin{aligned}
N_A &= \{\mathbf{ClientTimeout},\ \mathbf{SimpleServer_p},\ \mathbf{SimpleServer_s}\}\\
A &= \{(\mathbf{ClientTimeout},\ X_4,\ D_4,\ T_4^{>},\ T_4^{<}),\\
&\quad\ (\mathbf{SimpleServer_p},\ X_5,\ D_5,\ T_5^{>},\ T_5^{<}),\\
&\quad\ (\mathbf{SimpleServer_s},\ X_6,\ D_6,\ T_6^{>},\ T_6^{<})\}\\
N_T &= \{\mathbf{Proc_p},\ \mathbf{Proc_s}\}\\
T &= \{(\mathbf{Proc_p},\ \{\mathit{client},\ \mathit{server}_p\}),\ (\mathbf{Proc_s},\ \{\mathit{client},\ \mathit{server}_s\})\}\\
R &= \{\mathit{client},\ \mathit{server}_p,\ \mathit{server}_s\}\\
P &= p^u\\
S &= \{\mathbf{sent},\ \mathbf{processed}\}\\
V &= \{c,\ s\}
\end{aligned}$$

where:

$$
\begin{aligned}
X_4 &= \{T_{cp},\ T_{cs},\ \mathit{Send},\ \mathit{Wait},\ \mathit{Done},\ \mathit{Task}\} \\
D_4 &= \{(\mathit{Send},\ \mathit{Wait},\ \mathbf{t},\ \lambda,\ \emptyset,\ \{(s,\ \mathbf{sent})\}), \\
&\quad\ (\mathit{Wait},\ \mathit{Done},\ \mathbf{i},\ 1,\ \{(s,\ \mathbf{processed})\},\ \emptyset), \\
&\quad\ (\mathit{Done},\ \mathit{Task},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset), \\
&\quad\ (\mathit{Task},\ T_{cp},\ \mathbf{t},\ \mu,\ \emptyset,\ \emptyset), \\
&\quad\ (T_{cp},\ T_{cs},\ \mathbf{t},\ \nu,\ \emptyset,\ \emptyset)\} \\
T_4^{>} &= \{(T_{cp},\ \mathbf{Proc_p},\ \mathbf{client},\ \mathit{Send},\ \{(s,\ \mathit{server}_p)\}), \\
&\quad\ (T_{cs},\ \mathbf{Proc_s},\ \mathbf{client},\ \mathit{Send},\ \{(s,\ \mathit{server}_s)\})\} \\
T_4^{<} &= \{((\mathit{Done},\ \mathit{Task},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\ s)\}
\end{aligned}
$$

$$
\begin{aligned}
X_5 &= \{T_{sp},\ \mathit{Wait},\ \mathit{Process},\ \mathit{Done},\ \mathit{Reset}\} \\
D_5 &= \{(\mathit{Wait},\ \mathit{Process},\ \mathbf{i},\ 1,\ \{(c,\ \mathbf{sent})\},\ \emptyset), \\
&\quad\ (\mathit{Process},\ \mathit{Done},\ \mathbf{t},\ \gamma,\ \emptyset,\ \{(c,\ \mathbf{processed})\}), \\
&\quad\ (\mathit{Done},\ \mathit{Reset},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset), \\
&\quad\ (\mathit{Reset},\ T_{sp},\ \mathbf{t},\ \rho,\ \emptyset,\ \emptyset)\} \\
T_5^{>} &= \{(T_{sp},\ \mathbf{Proc_p},\ \mathbf{server_p},\ \mathit{Wait},\ \{(c,\ \mathit{client})\})\} \\
T_5^{<} &= \{((\mathit{Done},\ \mathit{Reset},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\ c)\}
\end{aligned}
$$

and:

$$
\begin{aligned}
X_6 &= \{T_{ss},\ \mathit{Wait},\ \mathit{Phase}_1,\ \mathit{Phase}_2,\ \mathit{Done},\ \mathit{Reset}\} \\
D_6 &= \{(\mathit{Wait},\ \mathit{Phase}_1,\ \mathbf{i},\ 1,\ \{(c,\ \mathbf{sent})\},\ \emptyset), \\
&\quad\ (\mathit{Phase}_1,\ \mathit{Phase}_2,\ \mathbf{t},\ \gamma_1,\ \emptyset,\ \emptyset), \\
&\quad\ (\mathit{Phase}_2,\ \mathit{Done},\ \mathbf{t},\ \gamma_2,\ \emptyset,\ \{(c,\ \mathbf{processed})\}), \\
&\quad\ (\mathit{Done},\ \mathit{Reset},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset), \\
&\quad\ (\mathit{Reset},\ T_{ss},\ \mathbf{t},\ \rho,\ \emptyset,\ \emptyset)\} \\
T_6^{>} &= \{(T_{ss},\ \mathbf{Proc_s},\ \mathbf{server_s},\ \mathit{Wait},\ \{(c,\ \mathit{client})\})\} \\
T_6^{<} &= \{((\mathit{Done},\ \mathit{Reset},\ \mathbf{i},\ 1,\ \emptyset,\ \emptyset),\ c)\}
\end{aligned}
$$