Performance Investigation of an On-Line Auction System [*]

Jane Hillston[1],   Leïla Kloul[2]

[1] LFCS
University of Edinburgh
Kings Buildings
Edinburgh EH9 3JZ, Scotland
email: jeh@dcs.ed.ac.uk

[2] PRiSM
Université de Versailles
45, Av. des Etats-Unis
78035 Versailles Cedex, France
email: kle@prism.uvsq.fr

Abstract

The standard design of on-line auction systems places most of the computational load on the server and its adjacent links, resulting in a bottleneck in the system. In this paper, we investigate the impact, in terms of the performance of the server and its adjacent links, of introducing active nodes into the network. The performance study of the system is done using the stochastic process algebra formalism PEPA.

1   Introduction

In this paper we investigate the interplay of two emerging technologies: *active networks* and software *agents* to support electronic commerce.

Active networks[1] are a compelling new initiative in networking. An active network extends a conventional one with the ability for network switches to process data as it is being transmitted. The processing which is to be performed can be customised by the network user on a per-application or even per-message basis. This innovation is a dramatic departure from traditional network design where the emphasis is on the avoidance of examination or modification of data. Active networks are supported by a variety of software technologies, execution environments and node operating systems [2].

Some recent work has focused on the performance gains that active network technology may bring to distributed applications [3]. Examples include active reliable multi-cast [4] and cache routing [5]. In [3] *on-line auctions* are suggested as applications which might benefit from processing at active nodes within the network.

The Internet offers exciting new prospects for electronic commerce, but it cannot always deliver the performance necessary to make them viable. The electronic transactions take a variety of forms but increasingly there is a move towards agent-based systems in which personalised, semi-autonomous software agents act on behalf of consumers or businesses [6]. In many cases such systems rely on the exchange of information and negotiation. If the integrity of such transactions is to be maintained, there is a clear need for timely behaviour of the underlying infrastructure.

Several on-line auction systems have been developed experimentally, such as the AuctionBot system (`auction.eecs.umich.edu`) from the University of Michigan, the Fishmarket Project (`www.fishmarket.com`) [7] or the eAuctionHouse (`ecommerce.cs.wustl.edu`) which supports combinatorial auctions and which is from the University of Washington. In such systems, competitive behaviour on the part of the bidder relies on a rapid response to submitted bids. However this may be jeopardised by network latency and/or server overload. As suggested, but not investigated in [3], the in-network processing capabilities provided by an active network appears to provide a solution to this problem. As far as we are aware no thorough performance analysis of such a scenario has been carried out.

In this paper, we investigate the performance issues surrounding such a situation. The idea we develop involves replacing standard, basic intermediary nodes of the network by active nodes, the goal being to transfer some tasks from the server to these nodes. This should result in a significant benefit in terms of both system throughput and system latency. The resulting system is then analysed using the stochastic process algebra modelling formalism PEPA [8].

The paper is organised as follows. In Section 2, we describe the on-line auction system we investigate, and the motivation for the approach that we take. Then, in Section 3, after a brief introduction to the modelling formalism we use, PEPA, we present the details of our model. Our solution technique is outlined in Section 4, together with the experiments we conducted and the numerical results we have obtained. Some conclusions of this work, together with the possible extensions, are discussed in Section 5.

2   The On-line Auction System

In an on-line auction system, a server receives and processes bids from remote software agents representing interested consumers. These semi-autonomous agents submit bids according to a predetermined strategy together with the information that they can ascertain from the server. The server processes bids, either accepting them

or rejecting them, depending on their value. In some systems additional attributes may be considered when comparing bids of the same or close value.

In addition to bids, bidder agents may also submit price notification requests, asking the server to tell them the latest bidding price. Note that the bidder agents can never be certain that they have an accurate representation of the current price due to network latency. They can, however, be certain that their current representation is out of date when a submitted bid, which is higher than their idea of the "current" price, is rejected.

The effectiveness of the bidder agents will depend on the proportion of time that their price information is accurate. Maintaining such accurate information places stringent performance requirements on the underlying infrastructure. Moreover the scalability of such systems, in terms of the number of bidder agents that can be satisfactorily accommodated, could be severely limited by the performance of the network. From the point of view of accessibility it is important that such auction systems use existing infrastructure, i.e. the Internet, and so the ability to directly address performance problems may be limited. However, in this paper we consider how such performance limitations may be circumvented, by incorporating active nodes within the network.
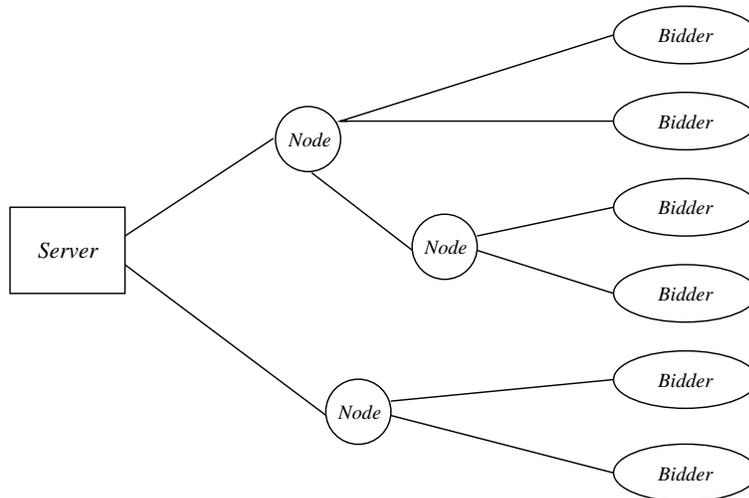

Figure 1: The on-line auction network topology

The standard design of an on-line auction system necessarily places most of the computational load on the server and its adjacent links, forming a bottleneck in the system. We investigate the advantages of introducing bid caches at intermediary nodes within the network between the server and some bidders (Figure 1). Such nodes are "active" in the sense that, in addition to routing, they examine the contents of bid and price request messages and form caches, storing recent bid and price information. These caches do not have the ability to accept bids but may act as filters, by rejecting bids which are known to be too low. This reduces the load on

3

both the server and its adjacent network elements. We assume that the cache's price information is gleaned from the bids which it handles together with their respective responses, but also from a periodic update message sent from the server. For bids which are passed to the server the latency due to processing within the network is increased; however the intention is that this will be more than compensated by the reduced traffic reaching the bottleneck of the system.

In order to quantify the advantages of introducing an active node to act as a cache for the on-line auction system, we compute and compare the performance of two such systems: one system deployed on a simple network without active nodes and another on the same network in which one node becomes active. The approach we take to develop performance measures is based on the stochastic process algebra PEPA (Performance Evaluation Process Algebra). PEPA serves as a high-level notation for Markov modelling: it is possible to automatically generate a continuous-time Markov process directly from the PEPA model which faithfully encodes the behavioural and temporal aspects of the modelled system. Details of this mapping can be found elsewhere [9]. Other high-level notations for Markov processes, such as GSPN [10] or SAN [11] could equally have been used but the compositional structure of PEPA seemed well-suited to the structure of the auction system. Using a formal model, such as a process algebra, allows us to additionally verify the functional correctness of the proposed system. Moreover, PEPA supports an automatic aggregation technique which allows the state space of the model to be reduced without loss of information, transparently to the user.

Our study is decomposed into two parts. In the first part, we consider the system with only traditional nodes as given by Figure 1. We model that system and compute its performance in terms of throughput. In the second part of the study, we replace one of the traditional nodes by an active one. In the models we adopt a number of assumptions and conventions.

- All bidder agents in the system adopt the same strategy. According to this strategy, whenever a bid is rejected the bidder submits a price notification request to get an updated price estimate before submitting any more bids.

- In the server, a serving agent is spawned to correspond with each bidder agent. This agent is responsible for maintaining the current state of interactions with the corresponding bidder.

- In the model all data dependent behaviour is abstracted. This means that we do not represent the current price, nor the value of a bid. Nor do we represent details of any bidding strategy, or selection strategy for choosing between bids of comparable value. Instead we use probabilities to represent the relative frequency with which bids are successful.

- Since bids which are subject to longer latencies are more likely to be unsuccessful, we adjust the acceptance probabilities according to the routes by which

4

bids will arrive at the server, or the cache.

- We make a distinction between the nodes of the system. This distinction reflects the position of the node in the system and thus the elements to which it is connected. However, note that the essential behaviour and timing characteristics of the nodes (without cache) are the same in each case.

- Our model is stochastic, meaning that all times are represented as random variables. Since we will use Markovian analysis to calculate performance measures, all random variables are assumed to be exponentially distributed. Performance measures are derived from equilibrium, or steady state, behaviour.

In the following section we briefly introduce the PEPA formalism, before presenting the models of the auction system in detail.

3   The PEPA models

PEPA (Performance Evaluation Process Algebra) extends classical process algebra by associating a random variable, representing duration, with every action. These random variables are assumed to be exponentially distributed giving a clear relationship between the process algebra model and a Markov process.

PEPA models are described as interactions of *components*. Each component can perform a set of actions: an action $a \in \mathcal{A}ct$ is described as a pair (α, r), where $\alpha \in \mathcal{A}$ is the *type* of the action and $r \in \mathbb{R}^+$ is the parameter of the negative exponential distribution governing its duration. Whenever a process P can perform an action, an instance of a given probability distribution is sampled: the resulting number specifies how long it will take to *complete* the action. A small but powerful set of combinators is used to build up complex behaviour from simpler behaviour. The combinators are familiar from classical process algebra: prefix, choice, parallel composition and abstraction. We explain each of the combinators informally below. A formal operational semantics for PEPA is available in [8].

Prefix: The prefix combinator "." is used to designate the first action in the behaviour of a component, e.g. $(\alpha, r).P$ will carry out an action of type α with an average duration of $1/r$ and then behave as component P. In some cases, the rate of an action is outside the control of this component. Such actions are carried out jointly with another component, with this component playing a passive role. In this case the rate of the action is denoted by the distinguished symbol, $\top$ (called "top").

Choice: A choice between two possible behaviours is represented as the sum of the possibilities, e.g. $(\alpha, r).P + (\beta, s).Q$. A race condition is assumed to govern the behaviour of simultaneously enabled actions so the choice combinator represents

5

pre-emptive selection with re-sampling. The continuous nature of the probability distributions ensures that the actions cannot occur simultaneously. Thus a sum will behave as one of its summands.

Parallel composition: Parallel composition is used to represent the cases when we expect two components of the system to *cooperate* to achieve some action. For example, the system $P \bowtie_L Q$ consists of two components P and Q which must cooperate to achieve actions which are in the *cooperation set L*. Actions with types not in this set may be carried out independently and concurrently by the two components. Actions in this set, *shared actions*, require the simultaneous involvement of both components. These will have the same type as the two contributing actions and a rate reflecting the rate in the slowest participating component. Note that this means that the rate of a passive action will become the rate of the action it cooperates with. When the cooperation set L is empty, we use the notation $P \parallel Q$ to denote independent concurrent behaviour.

Abstraction: It is often convenient to hide some actions, making them private to the component or components involved. The duration of the actions is unaffected, but their type becomes hidden, appearing instead as the unknown type τ. Components cannot synchronise on τ.

Using constants to name components and recursive definitions we are able to describe components with infinite behaviours without the use of an explicit recursion operator. Representing the components of the system as separate components means that we can easily extend our model.

3.1   The system without active node

The PEPA model corresponding to the on-line auction system with only traditional nodes is composed of six components. The configuration we consider is shown in Figure 2.

The PEPA model is composed of components $Server$, $Bidder$, $Node$ for the basic node, $CNode$ for the central node and $TNode$ for the upstream one. For technical reasons, a bidder connected to the central node is modelled using the component $Bidder_{CN}$. Action types are used to ensure the correct routing within the network model: the suffixes $\_csb$ and $\_csn$ are used to denote messages to the server via the upstream node and via the central node and the upstream node, respectively. The suffixes $\_scb$ and $\_scn$ denote messages in the reverse direction.

Let us now give details of the behaviour of the different components of the model.

Component $Server$**:** The auction server agent is represented by the PEPA component $Server$. In our configuration this component consists of six basic agents, responsible for each of the bidders. The behaviour of these agents are essentially
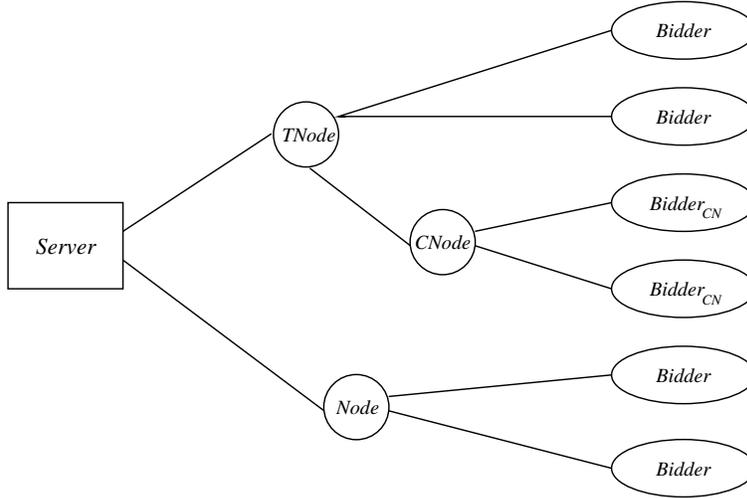
6

Figure 2: The On-line Auction System with PEPA Components

identical although the action types differ according to the route between the bidder and the server, as explained above. $ServerN$, $ServerC$ and $ServerT$, deal with the requests forwarded by the basic node, via the central node and directly through the upstream node, respectively. For example, the behaviour of $ServerN$ is that it waits to receive bids ($forward\_bid$) or price requests ($forward\_preq$) forwarded by the basic node. It is passive with respect to these actions. When receiving a bid, it (probabilistically) accepts or rejects that bid. On receipt of a price request it makes the price response action ($presp\_s$). In either case, we assume that the mean duration of the server's action is $1/r$. The multipliers p and q denote the different probabilities with which actions occur. The PEPA equations for the server's components are shown in Figure 3.

Component $Bidder$**:**   The software agent representing the potential buyer is represented by the PEPA component $Bidder$. The bidder submits a bid (bid_in) or a price request ($preq\_in$) via the network. It then enters a state waiting for an appropriate response. After a rejected bid, the bidder always submits a price request. After an accepted bid ($forward\_accept$) or a price response ($forward\_presp$) the bidder returns to its original state. We assume that the rate at which the bidder generates messages is s_3. In its basic state, $Bidder$, the component chooses between submitting a bid, at rate s_1, and submitting a price request at rate s_2. These rates reflect the relative probabilities of the two types of message and so we assume $s_1 + s_2 = s_3$. The PEPA equations for this component are shown in Figure 4.

The behaviour of a bidder attached to the central node ($CNode$) is essentially the same, although some actions have different names. The equations for such a bidder ($Bidder_{CN}$), are shown in Figure 5.

7

$$
\begin{aligned}
ServerN &\stackrel{\text{def}}{=} (forward\_bid, \top).ServerN' + (forward\_preq, \top).ServerN'' \\
ServerN' &\stackrel{\text{def}}{=} (accept\_s, p \times r).ServerN + (reject\_s, (1-p) \times r).ServerN \\
ServerN'' &\stackrel{\text{def}}{=} (presp\_s, r).ServerN
\end{aligned}
$$

$$
\begin{aligned}
ServerC &\stackrel{\text{def}}{=} (forward\_bid\_csn, \top).ServerC' + (forward\_preq\_csn, \top).ServerC'' \\
ServerC' &\stackrel{\text{def}}{=} (accept\_scn, q \times r).ServerC + (reject\_scn, (1-q) \times r).ServerC \\
ServerC'' &\stackrel{\text{def}}{=} (presp\_scn, r).ServerC
\end{aligned}
$$

$$
\begin{aligned}
ServerT &\stackrel{\text{def}}{=} (forward\_bid\_csb, \top).ServerT' + (forward\_preq\_csb, \top).ServerT'' \\
ServerT' &\stackrel{\text{def}}{=} (accept\_scb, p \times r).ServerT + (reject\_scb, (1-p) \times r).ServerT \\
ServerT'' &\stackrel{\text{def}}{=} (presp\_scb, r).ServerT
\end{aligned}
$$

$$
Server \stackrel{\text{def}}{=} ServerT \parallel ServerT \parallel ServerC \parallel ServerC \parallel ServerN \parallel ServerN
$$

Figure 3: PEPA definition of the server components

$$
\begin{aligned}
Bidder &\stackrel{\text{def}}{=} (bid\_in, s_1).WaitingBid + (preq\_in, s_2).WaitingPrice \\
WaitingBid &\stackrel{\text{def}}{=} (forward\_accept, \top).Bidder + (forward\_reject, \top).Incorrect \\
WaitingPrice &\stackrel{\text{def}}{=} (forward\_presp, \top).Bidder \\
Incorrect &\stackrel{\text{def}}{=} (preq\_in, s_3).WaitingPrice
\end{aligned}
$$

Figure 4: PEPA definition of the basic bidder component

Component $Node$**:** A basic node in the network is merely responsible for forwarding messages back and forth between the server and the bidders. It can passively accept messages of various types: $bid\_in, preq\_in, accept\_s, reject\_s, presp\_s$ corresponding to bids, price requests, bid acceptances, bid rejections and price responses respectively. It then forwards these appropriately, with rate s. The PEPA equations for this component are shown in Figure 6.

Component $CNode$**:** The central node has essentially the same functionality as the basic node. Some actions are renamed to avoid misrouting. The PEPA definitions for this type of node are shown in Figure 7.

Component $TNode$**:** The behaviour of the upstream node is similar to the behaviour of the basic node ($Node$) except that it may also have to forward bids and price requests that flow through the central node ($CNode$), distinguishing them from

8

$$
\begin{aligned}
Bidder_{CN} &\stackrel{\text{def}}{=} (bid'\_in, s_1).WaitingBid_{CN} + (preq'\_in, s_2).WaitingPrice_{CN} \\
WaitingBid_{CN} &\stackrel{\text{def}}{=} (forward\_accept', \top).Bidder_{CN} + (forward\_reject', \top).Incorrect_{CN} \\
WaitingPrice_{CN} &\stackrel{\text{def}}{=} (forward\_presp', \top).Bidder_{CN} \\
Incorrect_{CN} &\stackrel{\text{def}}{=} (preq'\_in, s_3).WaitingPrice_{CN}
\end{aligned}
$$

Figure 5: PEPA definition of a bidder attached to the central node ($CNode$)

$$
\begin{aligned}
Node &\stackrel{\text{def}}{=} (bid\_in, \top).Node_1 + (preq\_in, \top).Node_2 \\
&\quad + (accept\_s, \top).(forward\_accept, s).Node \\
&\quad + (reject\_s, \top).(forward\_reject, s).Node \\
&\quad + (presp\_s, \top).(forward\_presp, s).Node \\
Node_1 &\stackrel{\text{def}}{=} (forward\_bid, s).Node \\
&\quad + (accept\_s, \top).(forward\_accept, s).Node_1 \\
&\quad + (reject\_s, \top).(forward\_reject, s).Node_1 \\
&\quad + (presp\_s, \top).(forward\_presp, s).Node_1 \\
Node_2 &\stackrel{\text{def}}{=} (forward\_preq, s).Node \\
&\quad + (accept\_s, \top).(forward\_accept, s).Node_2 \\
&\quad + (reject\_s, \top).(forward\_reject, s).Node_2 \\
&\quad + (presp\_s, \top).(forward\_presp, s).Node_2
\end{aligned}
$$

Figure 6: PEPA definition of the basic node

those that it receives from a bidder agent directly. For that reason, the component $TNode$ consists of two independent components $TNodeB$ and $TNodeN$. Thus the actions corresponding to messages routed between the $CNode$ and the $TNode$ and vice versa have suffix _nc_ and _sc_ respectively. As previously, receiving messages is assumed to be passive, whilst transmitting messages occurs at rate s. The defining equations for the corresponding PEPA components are shown in Figure 8.

The Complete System   The complete model has 87480 states and 405864 transitions after automatic aggregation. The PEPA equation for our configuration is the following:

$$((Server \underset{L_1}{\bowtie} (Bidders \underset{L_2}{\bowtie} TNode) \underset{L_3}{\bowtie} (CNode \underset{L_4}{\bowtie} Bidders_{CN})) \underset{L_5}{\bowtie} (Node \underset{L_6}{\bowtie} Bidders)$$

where

$$Bidders \stackrel{\text{def}}{=} Bidder \parallel Bidder \qquad \text{and} \qquad Bidders_{CN} \stackrel{\text{def}}{=} Bidder_{CN} \parallel Bidder_{CN}$$

$$
\begin{aligned}
CNode \ &\stackrel{\text{def}}{=}\ (bid'\_in, \top).CNode_1 + (preq'\_in, \top).CNode_2 \\
&\quad + (forward\_accept\_sc, \top).(forward\_accept', s).CNode \\
&\quad + (forward\_reject\_sc, \top).(forward\_reject', s).CNode \\
&\quad + (forward\_presp\_sc, \top).CNode_5 \\
CNode_1 \ &\stackrel{\text{def}}{=}\ (forward\_bid\_nc, s).CNode \\
&\quad + (forward\_accept\_sc, \top).(forward\_bid\_nc, s).CNode_3 \\
&\quad + (forward\_reject\_sc, \top).(forward\_bid\_nc, s).CNode_4 \\
&\quad + (forward\_presp\_sc, \top).(forward\_bid\_nc, s).CNode_5 \\
CNode_2 \ &\stackrel{\text{def}}{=}\ (forward\_preq\_nc, s).CNode \\
&\quad + (forward\_accept\_sc, \top).(forward\_preq\_nc, s).CNode_3 \\
&\quad + (forward\_reject\_sc, \top).(forward\_preq\_nc, s).CNode_4 \\
&\quad + (forward\_presp\_sc, \top).(forward\_preq\_nc, s).CNode_5 \\
CNode_3 \ &\stackrel{\text{def}}{=}\ (forward\_accept', s).CNode \\
CNode_4 \ &\stackrel{\text{def}}{=}\ (forward\_reject', s).CNode \\
CNode_5 \ &\stackrel{\text{def}}{=}\ (forward\_presp', s).CNode
\end{aligned}
$$

Figure 7: PEPA definition of the central node

The cooperation sets are defined as follows:

$$
\begin{aligned}
L_1 \ &=\ \{accept\_scb, reject\_scb, presp\_scb, accept\_scn, reject\_scn, presp\_scn, \\
&\qquad forward\_bid\_csb, forward\_preq\_csb, forward\_bid\_csn, forward\_preq\_csn\} \\
L_2 \ &=\ \{bid\_in, preq\_in, forward\_accept, forward\_reject, forward\_presp\} \\
L_3 \ &=\ \{forward\_bid\_nc, forward\_preq\_nc, forward\_accept\_sc, \\
&\qquad forward\_reject\_sc, forward\_presp\_sc\} \\
L_4 \ &=\ \{bid'\_in, preq'\_in, forward\_accept', forward\_reject', forward\_presp'\} \\
L_5 \ &=\ \{accept\_s, reject\_s, presp\_s, forward\_preq, forward\_bid\} \\
L_6 \ &=\ \{bid\_in, preq\_in, forward\_accept, forward\_reject, forward\_presp\}
\end{aligned}
$$

3.2 The system with an active node

Consider now the same configuration but with one of the nodes (the upstream one) replaced by an active one. This node is active in the sense that, in addition to routing, it examines the contents of bid and price request messages that flow through it. Because of the cache it provides, this node may store bid and price information. Moreover this allows the active node to filter the messages to the server. The corresponding PEPA model of such a system differs from the cacheless model essentially by the component *Cache* which models the active node and which

$$TNodeB \quad \stackrel{\mathrm{def}}{=} \quad (bid\_in, \top).(forward\_bid\_csb, s).TNodeB$$
$$+ \quad (preq\_in, \top).(forward\_preq\_csb, s).TNodeB$$
$$+ \quad (accept\_scb, \top).(forward\_accept, s).TNodeB$$
$$+ \quad (reject\_scb, \top).(forward\_reject, s).TNodeB$$
$$+ \quad (presp\_scb, \top).(forward\_presp, s).TNodeB$$

$$TNodeN \quad \stackrel{\mathrm{def}}{=} \quad (forward\_bid\_nc, \top).(forward\_bid\_csn, s).TNodeN$$
$$+ \quad (forward\_preq\_nc, \top).(forward\_preq\_csn, s).TNodeN$$
$$+ \quad (accept\_scn, \top).(forward\_accept\_sc, s).TNodeN$$
$$+ \quad (reject\_scn, \top).(forward\_reject\_sc, s).TNodeN$$
$$+ \quad (presp\_scn, \top).(forward\_presp\_sc, s).TNodeN$$

$$TNode \quad \stackrel{\mathrm{def}}{=} \quad TNodeB \parallel TNodeN$$

Figure 8: PEPA definition of the upstream node

replaces component $TNode$ (Figure 9). Otherwise, we have the same components: $Server$, $Bidder$, $Bidder_{CN}$, $Node$ and $CNode$. Indeed, some of these, $Bidder_{CN}$ and $Node$ remain completely unchanged.
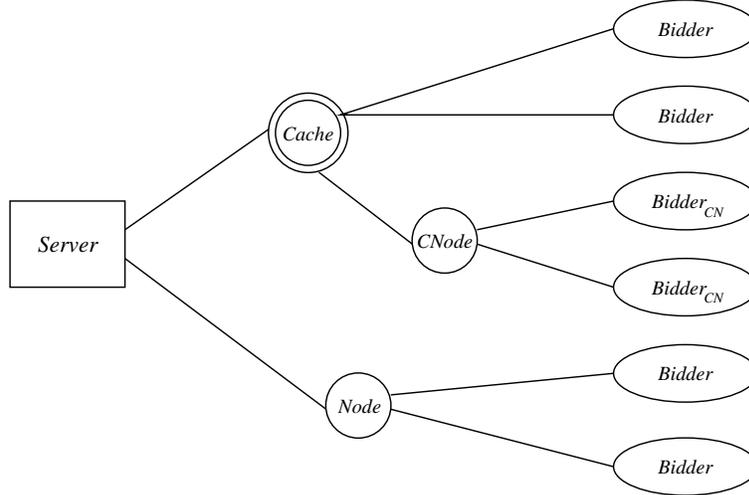

Figure 9: The On-line Auction system with PEPA components

Component *Server*: The server agents retain much of their behaviour from the cacheless model; only the handling of price requests is removed from $ServerT$ and

ServerC agents since these messages are now filtered by the cache. Moreover, as the server has to send periodic messages to update the price information in the active node, we need to introduce a new action (*update*) on which these agents and the cache must synchronise. As previously, different probabilities reflect the different expected latencies of bids arriving by different routes. Due to the increased processing at the active node, the latency of bids arriving at *ServerN* and *ServerT* can no longer be regarded as the same. The new PEPA equations for this component are shown in Figure 10.

$$
\begin{aligned}
ServerT &\stackrel{\text{def}}{=} (forward\_bid\_csb, \top).ServerT' + (update, w).ServerT \\
ServerT' &\stackrel{\text{def}}{=} (accept\_scb, p_1 \times r).ServerT + (reject\_scb, (1 - p1) \times r).ServerT
\end{aligned}
$$

$$
\begin{aligned}
ServerC &\stackrel{\text{def}}{=} (forward\_bid\_csn, \top).ServerC' + (update, w).ServerC \\
ServerC' &\stackrel{\text{def}}{=} (accept\_scn, p_2 \times r).ServerC + (reject\_scn, (1 - p_2) \times r).ServerC
\end{aligned}
$$

$$
\begin{aligned}
ServerN &\stackrel{\text{def}}{=} (forward\_bid, \top).ServerN' + (forward\_preq, \top).ServerN'' \\
ServerN' &\stackrel{\text{def}}{=} (accept\_s, p_3 \times r).ServerN + (reject\_s, (1 - p_3) \times r).ServerN \\
ServerN'' &\stackrel{\text{def}}{=} (presp\_s, r).ServerN
\end{aligned}
$$

$$
Server \stackrel{\text{def}}{=} (ServerT \underset{\{update\}}{\bowtie} ServerT \underset{\{update\}}{\bowtie} ServerC \underset{\{update\}}{\bowtie} ServerC) \parallel ServerN \parallel ServerN
$$

Figure 10: PEPA definition of the server component

Component *Bidder*: The introduction of the cache means that the bidders connected to the active node may receive two new message types, *presp_c* and *reject_c*, representing price responses and bid rejections respectively. These are synchronised with the cache. Conversely, price responses are no longer expected from the server. The PEPA equations for this component are shown in Figure 11.

$$
\begin{aligned}
Bidder &\stackrel{\text{def}}{=} (bid\_in, s_1).WaitingBid + (preq\_in, s_2).WaitingPrice \\
WaitingBid &\stackrel{\text{def}}{=} (forward\_accept, \top).Bidder + (forward\_reject, \top).Incorrect \\
&\quad + (reject\_c, \top).Incorrect \\
WaitingPrice &\stackrel{\text{def}}{=} (presp\_c, \top).Bidder \\
Incorrect &\stackrel{\text{def}}{=} (preq\_in, s_3).WaitingPrice
\end{aligned}
$$

Figure 11: PEPA definition of the basic bidder component

Component $CNode$**:**   At the central node, the action representing price responses from the server ($forward\_presp\_sc$) is replaced by one representing a response from the cache ($presp\_cn$). We also introduce a new action representing the rejection of bids by the cache: $reject\_cn$. The PEPA definitions for this type of node are shown in Figure 12.

$$
\begin{aligned}
CNode \;\stackrel{\text{def}}{=}\; & (bid'\_in, \top).CNode_1 + (preq'\_in, \top).CNode_2 \\
& + (forward\_accept\_sc, \top).(forward\_accept', s).CNode \\
& + (forward\_reject\_sc, \top).(forward\_reject', s).CNode \\
& + (reject\_cn, \top).(forward\_reject', s).CNode \\
& + (presp\_cn, \top).(forward\_presp', s).CNode \\[4pt]
CNode_1 \;\stackrel{\text{def}}{=}\; & (forward\_bid\_nc, s).CNode \\
& + (forward\_accept\_sc, \top).(forward\_accept', s).CNode_1 \\
& + (forward\_reject\_sc, \top).(forward\_reject', s).CNode_1 \\
& + (reject\_cn, \top).(forward\_reject', s).CNode_1 \\
& + (presp\_cn, \top).(forward\_presp', s).CNode_1 \\[4pt]
CNode_2 \;\stackrel{\text{def}}{=}\; & (forward\_preq\_nc, s).CNode \\
& + (forward\_accept\_sc, \top).(forward\_accept', s).CNode_2 \\
& + (forward\_reject\_sc, \top).(forward\_reject', s).CNode_2 \\
& + (reject\_cn, \top).(forward\_reject', s).CNode_2 \\
& + (presp\_cn, \top).(forward\_presp', s).CNode_2
\end{aligned}
$$

Figure 12: PEPA definition of the $CNode$ in the modified system

Component $Cache$**:**   The $Cache$ component, based on the $TNode$ component of the cacheless model, reflects the additional capabilities of the active node, i.e. to intercept bids and price requests and generate responses itself. On receiving a bid it will examine it and either reject it immediately ($reject\_c$ or $reject\_cn$) or pass it on to the server ($forward\_bid\_csb$ or $forward\_bid\_csn$). On receiving a price request it will respond to it directly ($presp\_c$ or $presp\_cn$). On its own behalf it may receive $update$ messages from the server. When handling bids or price requests the active node is expected to have timing behaviour similar to that of the server, so the rate of processing these messages is now r instead of s. The defining equations for the corresponding PEPA components are shown in Figure 13.

The probabilities q_1 and q_2 reflect the proportion of bids which are filtered out by the active node because the bid value is known to be too low. These probabilities are assumed to differ because the latency will be higher for the messages which arrive via the intermediate node.

13

$$
\begin{aligned}
CacheB \;&\overset{\text{def}}{=}\; (bid\_in, \top).CacheB_1 + (preq\_in, \top).CacheB_2 + (update, \top).CacheB \\
&\quad + (accept\_scb, \top).CacheB_3 + (reject\_scb, \top).CacheB_4 \\
CacheB_1 \;&\overset{\text{def}}{=}\; (forward\_bid\_csb, (1-q_1) \times r).CacheB + (reject\_c, q_1 \times r).CacheB \\
CacheB_2 \;&\overset{\text{def}}{=}\; (presp\_c, r).CacheB \\
CacheB_3 \;&\overset{\text{def}}{=}\; (forward\_accept, s).CacheB \\
CacheB_4 \;&\overset{\text{def}}{=}\; (forward\_reject, s).CacheB
\end{aligned}
$$

$$
\begin{aligned}
CacheN \;&\overset{\text{def}}{=}\; (forward\_bid\_nc, \top).CacheN_1 + (forward\_preq\_nc, \top).CacheN_2 \\
&\quad + (accept\_scn, \top).CacheN_3 + (reject\_scn, \top).CacheN_4 \\
&\quad + (update, \top).CacheN \\
CacheN_1 \;&\overset{\text{def}}{=}\; (forward\_bid\_csn, (1-q_2) \times r).CacheN + (reject\_cn, q_2 \times r).CacheN \\
CacheN_2 \;&\overset{\text{def}}{=}\; (presp\_cn, r).CacheN \\
CacheN_3 \;&\overset{\text{def}}{=}\; (forward\_accept\_sc, s).CacheN \\
CacheN_4 \;&\overset{\text{def}}{=}\; (forward\_reject\_sc, s).CacheN
\end{aligned}
$$

$$
Cache \;\overset{\text{def}}{=}\; CacheB \parallel CacheN
$$

Figure 13: PEPA definition of an active node (Cache)

The complete system: The PEPA equation for the configuration depicted in Figure 9 is shown below. This model has 41472 states and 222588 transitions after automatic aggregation.

$$
((Server \underset{L_1'}{\bowtie} (Bidders \underset{L_2'}{\bowtie} Cache) \underset{L_3'}{\bowtie} (CNode \underset{L_4}{\bowtie} Bidders_{CN})) \underset{L_5}{\bowtie} (Node \underset{L_6'}{\bowtie} Bidders)
$$

where L_4 and L_5 are the action sets defined in the basic model, and where

$$
\begin{aligned}
L_1' \;&=\; \{accept\_scb, reject\_scb, forward\_bid\_csb, accept\_scn, reject\_scn, \\
&\qquad forward\_bid\_csn, update\} \\
L_2' \;&=\; \{bid\_in, preq\_in, forward\_accept, forward\_reject, presp\_c, \\
&\qquad reject\_c, forward\_presp\} \\
L_3' \;&=\; \{forward\_bid\_nc, forward\_accept\_sc, forward\_preq\_nc, \\
&\qquad forward\_reject\_sc, presp\_cn, reject\_cn\} \\
L_6' \;&=\; \{bid\_in, preq\_in, forward\_accept, forward\_reject, forward\_presp, \\
&\qquad presp\_c, reject\_c\}
\end{aligned}
$$

14

4 Experiments and Numerical Results

Through the analysis and solution of the Markov process underlying a PEPA model, the modeller can undertake an experimental investigation of the system. The PEPA Workbench is a suite of tools which perform the well-formedness checking of PEPA models as well as the generation and solution of the corresponding Markov process [9]. It detects faults such as deadlocks and cooperations which do not involve active participants. In the most recent version, it includes support for a modal logic, allowing behavioural requirements of a model to be formally expressed and automatically checked [12].

In essence, the translation process which occurs within the PEPA Workbench accepts a PEPA model as input and produces a matrix containing the Markov process encoding of the model given. In the most straightforward translation a state of the Markov process is associated with each syntactic term of the PEPA model obtained by application of the structured operational semantics rules. To solve the models presented in this paper we took advantage of a modified version of the Workbench which automatically aggregates models, during exploration of the state space. Using this translation one state of the Markov process is associated with each equivalence class of states, where two states are considered equivalent if they generate the same observable behaviour. More details of this automatic aggregation can be found in [13].

Performance measures are derived via the steady state probability distribution of the Markov process. A variety of linear algebra techniques may be employed to obtain this vector and the PEPA Workbench supports a number of them. To solve the models presented here we used the preconditioned biconjugate gradient method. This is implemented as a C program and is the most efficient of the available solvers.

We conducted two sets of model solutions. In the first experiment we compared the cacheless system and the cached system, under varying workloads. For each model solution we calculated the average throughput of the server. In the case of the cached system, we also computed the cache throughput. In the final experiment we considered only the cached system and investigated the effect of varying the reject probability of the bids at the cache level. In all cases we made the following assumptions:

- The workload is uniformly split between the bidders, that is they all generate bids and respond to returned bids at the same rate. This rate is parameter s_3 which is varied during the experiments between the values 0.1 and 10. Moreover, we assume a proportion of 4 bids for 1 price request i.e. $s_1 = 4 \times s_2$.
- If the route of a bid through the network is longer, resulting in greater latency, the probability of the bid being successful is less, i.e. in the original $Server$, $q < p$.
- The probability p, in the cacheless model, reflects the percentage of successful bids arriving by the route ($Bidder \rightarrow TNode/Cache \rightarrow Server$). This

15

Rate	Value	Rate	Value
p	0.7	w	2.0
q	0.4	q_1	0.2
r	2.0	q_2	0.5
s	5.0		

Table 1: Input parameters

percentage is assumed to remain fixed when the active node is acting as fil-
ter for the bids. For this route, the probability that a bid is passed from
the cache to the server is $(1 - q_1)$. The probability that the bid is ac-
cepted by the server is then set as $p_1 = p/(1 - q_1)$. Similarly for the route
$(Bidder \rightarrow CNode \rightarrow TNode/Cache \rightarrow Server)$, $p_2 = q/(1 - q_2)$, whilst for
the route $(Bidder \rightarrow Node \rightarrow Server)$ the probability that a bid is accepted
remains unchanged $p = p_3$.

- Throughout the experiments the basic rates of processing by nodes and the
  server remain unchanged: non-active nodes process messages at rate s and the
  server agent processes messages at rate r, where we assume that $s > r$. The
  active node, which mimics the behaviour of the server, processes messages at
  rate r.

Our primary objective was to evaluate the impact of introducing an active node
(cache) on the server load; the performance criteria we are interested in are server
throughput and the cache throughput. As there are no losses in our system, evaluat-
ing these two measures will allow us to estimate the proportion of messages entering
the system which are handled solely by the cache. The server throughput is mea-
sured in terms of bids and price responses and the cache throughput in terms of
price responses and bids rejected at the cache level. The values of the rates we have
used in the experiments are given in Table 1.

In the first experiment, we study the impact of the active node on the server
load. Figures 14 to 16 summarise the main results we have obtained.

Figure 14 shows the behaviour of the server throughput in both models as a
function of the message request arrival rate (s_3). We can see that as the arrival rate
increases, both throughtputs increase. In the cacheless model, the throughput of
the server represents also the throughput of the system.

We can observe in that figure that the difference beween the two curves increases
as the arrival rate increases. This difference corresponds to the throughput that the
cache may have in the cached model. But when comparing the system throughput
of the cacheless model with the total throughput of the cached model (Server +
Cache), we obtain the results depicted by Figure 15. These results show that the
total throughput is reduced by the introduction of the active node.
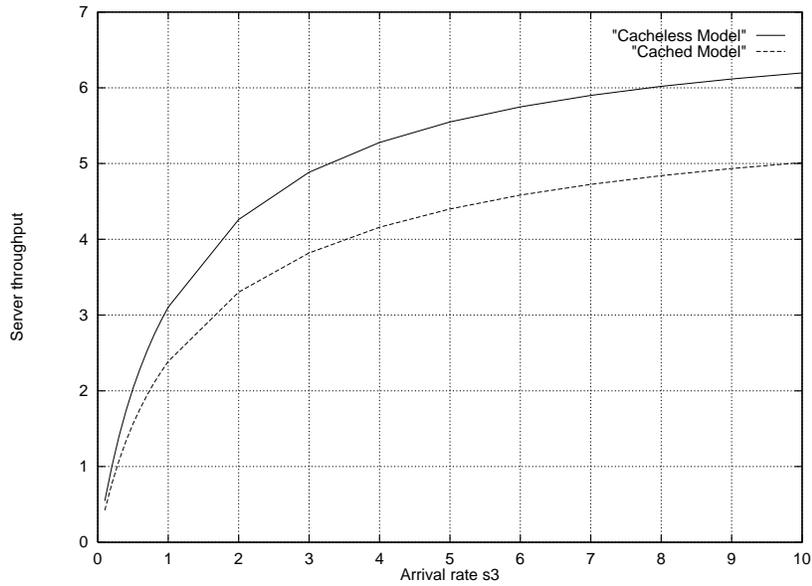
16
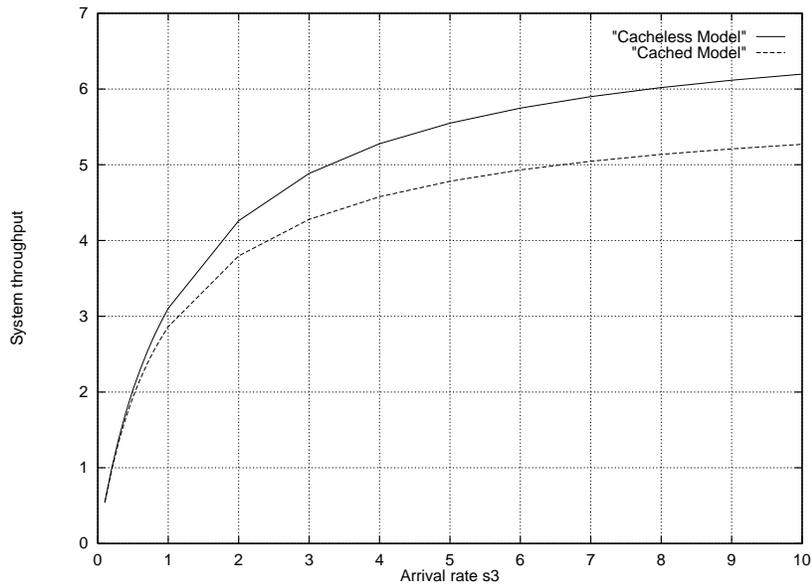
Figure 14: Server throughput versus arrival rate

Figure 15: Total system throughput versus arrival rate

Even though the cache throughput increases, Figure 16 shows that the proportion of the system throughput that it represents decreases. This figure consists of two curves. The first one depicts the proportion of the system throughput handled by the server and the second one depicts the proportion of the system throughput handled by the cache.

As the arrival rate s_3 increases, the first one increases whereas the second one
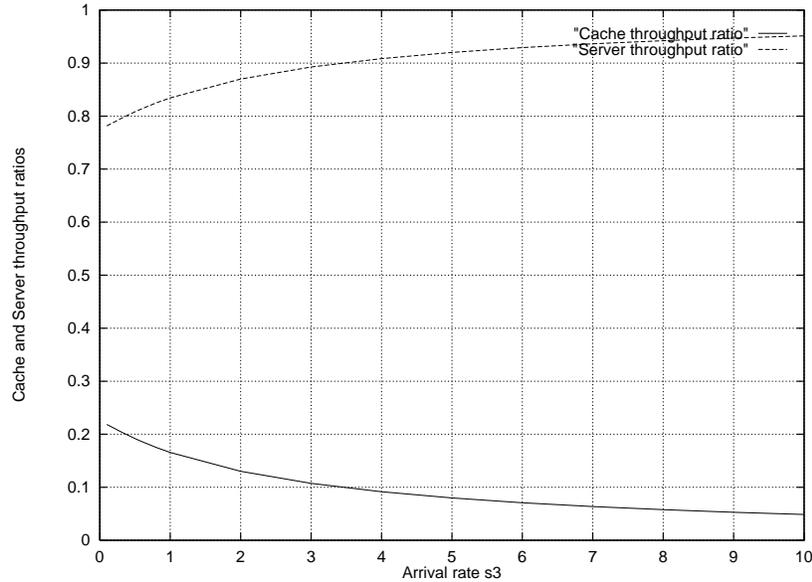
Figure 16: Cache and server throughput ratios versus arrival rate

decreases. This might suggest that the cache becomes a bottleneck in the system. Thus the bidders attached via the cache spend a significant proportion of their time blocked, waiting for a response. In contrast, the bidders attached to the server only via the non-active node spend relatively more time generating new messages, all of which are handled by the server. However, as discussed below, Figure 17, makes it clear that the server remains the bottleneck within the system. Nevertheless, the increased latency experienced by bidders attached via the cache reduces the frequency with which they can submit messages, and consequently the throughput of the cache. This has the effect of making the system throughput decrease and the proportion of the server throughput increase. For the cache, as the arrival rate increases, the throughput also increases (see Figure 14), but not enough to make the proportion of the system throughput it represents increase. Note, however, that more than 20% of requests may be treated and satisfied at the cache level.

The objective of the second experiment was to study the impact of the values of the probabilities q_1 and q_2 on the cache throughput. These probabilities represent the relative frequency with which the submitted bids are rejected by the cache. Thus, the higher this frequency is, the lower the proportion of bids forwarded to the server will be.

Figure 17 depicts the behaviour of the cache throughput and the influence of the reject probabilities on this performance measure. The first curve corresponds to the case when the reject probabilities are relatively high ($q_1 = 0.6$ and $q_2 = 0.8$) and the second to the case when they are relatively low ($q_1 = 0.2$ and $q_2 = 0.5$). In both curves and as the arrival rate increases, the cache throughput increases until a
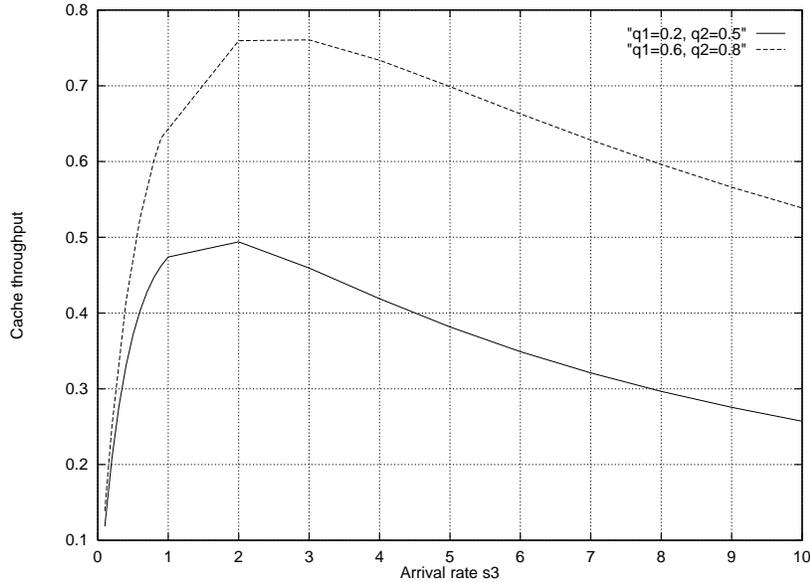
18

Figure 17: Cache throughput versus arrival rate

certain point and then decreases slowly. Note that in the case of high probabilities, the throughput is higher and it begins to decrease later.

In both cases, the decrease of the throughput may be explained by the limiting effect of the server as it becomes overloaded. This effect is less pronounced in the case when the probability of a bid being rejected by the cache is relatively high. Moreover, when the server is overloaded the number of bids generated by the bidders decreases as each one spends a greater proportion of its time waiting for responses.

To summarise, we believe that the cache throughput plays an increasingly important role as cache's ability to reject bids increases. In the system we investigated, the cache has a high service demand since more than 65% of the bidders are connected to it directly or via another node. The results we obtained suggest that if we consider another system topology, such as a system with one cache for each pair of bidders, the contribution of the cache would be more significant. Using several caches means using several filters in the system and dividing the total service demand of the bidders. This will certainly reduce the latency experienced by the bidders, especially if the relative frequency at which the submitted bids are rejected by the cache is high and the price request rate is significant. Otherwise, as stated previously, the real bottleneck remains the server.

5   Conclusion

In this paper, we have investigated an on-line auction system using the process algebra formalism PEPA. In this study, we were interested in the impact on the server

19

load of introducing active nodes as caches in intermediary nodes. Our performance measures have concerned the cache throughput and the system throughput, and the relation between them.

The system topology considered is rather simple, a server, three nodes, of which one is active, and six bidders. However this has allowed us to have an idea about the ratio of load the cache may treat. Further performance measures, such as system latency, remain to be studied in the future. An extension of this work consists of considering a more realistic topology with several active nodes and a greater number of bidder agents.

Another future work is to investigate placement strategies for active nodes in the network. Indeed, the effectiveness of the caches to filter unsuccessful bids relies on them capturing such bids as soon as possible. This would seem to suggest that they should be placed towards the edges of the network, close to the bidders. On the other hand, their ability to act as a filter relies on them having up-to-date price information and being exposed to as many bids as possible. This would seem to suggest that they should be placed close to the server.

References

[1] D.L. Tennenhouse and D. Wetherall. Towards an active network architecture. In *Multimedia Computing and Networking 96*, San Jose, January 1996.

[2] J.M. Smith, K.L. Calvert, S.L. Murphy, H.K. Orman, and L.L. Peterson. Activating Networks: A Progress Report. *IEEE Computer*, (4):32–41, April 1999.

[3] U. Legedza, D. Wetherall, and J. Guttag. Improving the performance of distributed applications using active networks. In *IEEE INFOCOM*, San Francisco, January 1998. IEEE Computer Society Press.

[4] L.H. Leiman, S.J.Garland, and D.L. Tennenhouse. Active reliable multicast. In *IEEE INFOCOM*, San Francisco, January 1998. IEEE Computer Society Press.

[5] U. Legedza and J. Guttag. Using network-level support to improve cache routing. In *Proc. 3rd Int. WWW Caching Workshop*, Manchester, England, June 1998.

[6] P. Maes R.H. Guttman A.G. Moukas. Agents that Buy and Sell. *Communications of the ACM*, 42(3):81–91, 1999.

[7] J. Rodriquez, P. Noriega, C. Sierra, and J. Padget. Fm96.5: A java-based electronic auction house. In *Proc. of 2nd Int. Conf. on Practical Application of Intelligent Agents and Multi-agent Technology (PAAM'97)*, London, April 1997.

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

[9] S. Gilmore and J. Hillston. The PEPA Workbench: A tool to support a process algebra based approach to performance modelling. In *Proc. 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in Lecture Notes in Computer Science, pages 353–368, Vienna, May 1994. Springer-Verlag.

[10] M. Ajmone Marsan, G. Conte, and G. Balbo. A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.

[11] B. Plateau. *De l'évaluation du parallélisme et de sa synchronisation*. PhD thesis, Université de Paris-Sud, Centre d'Orsay, 1984.

[12] G. Clark, S. Gilmore, J. Hillston, and M. Ribaudo. Exploiting modal logic to express performance measures. To appear in Performance Tools 2000, the Eleventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Illinois, USA, March 2000.

[13] S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. *IEEE Transactions on Software Engineering*, to appear, 2000.