Performance Modelling and Evaluation of Event-Condition-Action Rules on RDF in P2P networks

George Papamarkos, Alexandra Poulovassilis, Peter T. Wood
g.papamarkos@dcs.bbk.ac.uk, ap@dcs.bbk.ac.uk, ptw@dcs.bbk.ac.uk

Abstract

This paper studies the performance and scalability aspects of processing Event-Condition-Action (ECA) rules on RDF metadata in peer-to-peer (P2P) environments. This work has been motivated by the increasing use of RDF in distributed web-based applications that require timely notification and propagation of metadata changes between peers, for which ECA rules are a possible candidate technology.

We develop a performance model for analysing the behaviour of a rule processing system for ECA rules on RDF metadata in P2P environments. We use as the main performance criterion the time required to complete all rule processing resulting from a top-level update submitted to one of the peers in the network. We examine how this time varies with the network topology, number of peers, number of rules, and degree of metadata replication between peers. We describe a simulation of the system, and the analytical results are compared with analogous experiments conducted with the simulation, which show good agreement.

Our overall conclusions are that if a HyperCup topology is used for interconnecting the superpeers, then ECA rule processing on RDF metadata in P2P environments shows good scalability, pointing to its practical usefulness as a technology for real applications.

1 Introduction

RDF [35, 37] is one of the technologies proposed to realise the vision of the Semantic Web, and it is being increasingly used in distributed web-based applications, particularly in areas such as e-science and e-learning. Such applications may need to be *reactive*, i.e. to be able to detect the occurrence of specific events or changes in the RDF descriptions, and to respond by automatically executing the appropriate application logic.

Event-condition-action (ECA) rules are one way of providing this kind of functionality. An ECA rule has the general syntax `on` *event* `if` *condition* `do` *actions*. The event part specifies when the rule is *triggered*. The condition part is a query which determines if the information system is in a particular state, in which case the rule *fires*. The action part states the actions to be performed if the rule fires. These actions may in turn cause further events to occur, which may in turn cause more ECA rules to fire. We refer the reader to [38, 25] for a general discussion of ECA rules (*triggers*) in databases, where they are used for activities such as incremental maintenance of materialised views and replicas, constraint enforcement, and maintaining audit trails and database usage statistics. More broadly, ECA rules are also used in workflow management, network management, personalisation, and publish/subscribe technology. In the applications that we envisage, the rules are likely not to be hand-crafted but automatically generated by higher-level presentation and application services.

In our previous work [14] we have defined an ECA language for RDF called RDFTL, an architecture for supporting RDFTL rules in P2P environments, and an implementation of this architecture. The aim of this paper is to study the performance and scalability of this approach, and hence determine its potential as a practically useful technology for real applications. Microsoft's recently announced Simple Sharing Extensions (SSE) specification [30] enables the creation of bi-directional RSS feeds and is concerned with maintaining mutual consistency among these feeds. Such application areas provide further opportunities for applying ECA rules in distributed, collaborative environments where the automatic update and synchronisation of shared and replicated RDF resources is of principal concern.

Section 2 of the paper gives an overview of related work in ECA rules, performance modelling and P2P systems. Section 3 describes the RDFTL language, and the architecture and prototype implementation of the system supporting RDFTL rules in P2P environments. Section 4 develops an analytical performance model for this system, taking as the main performance criterion the average time that is needed to complete all rule execution resulting from a single update transaction that is submitted to the system by a user. Section 5 presents experimental results from the analytical study, which examines how this average time varies with the network topology, number of peers, number of rules, and degree of metadata replication between peers. Section 5 also describes a simulation of the system and presents analogous experimental results with this simulator. Section 6 gives our conclusions and a discussion of possible future work.

2 Related Work

To our knowledge, RDFTL (RDF Triggering Language) [14] is the first ECA rule language developed specifically for RDF/S. In particular, [14] discusses the syntax and execution semantics of the language, an architecture supporting such rules in P2P environments, and an implementation of this architecture. It also discusses techniques for relaxing the isolation and atomicity requirements of transactions in P2P environments. Languages for updating RDF descriptions have been considered in [22, 21]. The Modification Exchange Language (MEL) of [22] is based on an RDF representation of Datalog, and is used for updating RDF in the distributed environment of Edutella [23]. RUL (RDF Update Language) [21] is based on the RQL [19] query language and provides update functionality for RDFSuite [16].

There have been several languages proposed for ECA rules on XML data [7, 4, 3, 8, 6]. Active XML [1] provides similar functionality to that provided by XML ECA rules by embedding calls to web services within XML documents via special tags, aiming to integrate distributed data and distributed computation in P2P architectures. A model for distributing and replicating Active XML documents is presented in [2]. In order to facilitate distributed query evaluation, a location-aware extension of XPath and XQuery is described. A replication algorithm provides recommendations regarding what and where to replicate data in order to improve each peer's query performance. This model of P2P data replication differs from that of RDFTL in that the latter includes no explicit references to locations within ECA rules, but instead the replication results implicitly from the execution of ECA rule actions at all peers on which these updates can successfully be applied.

Other related work includes [18] which discusses using ECA rules in P2P systems in order to encode policies for the exchange of data between the peer databases. ECA rules have also been used to build Web applications that adapt to navigation patterns displayed by the

user [10].

Previous work on modelling and performance evaluation of active rule processing systems has concentrated mostly on benchmarking and simulation techniques rather than on analytical methods. The BEAST benchmark is employed in [15] in order to evaluate the performance of event detection, rule management and rule execution in SAMOS [13]. Tests are defined for each component focusing on the time required to complete a processing step, detect events of various types, retrieve the correct rule from the rule base and execute rules in various coupling modes.

A set of simulation experiments are performed in [5] to evaluate the performance trade-offs for different rule execution semantics. The average transaction response time, defined as the time elapsed from the transaction's arrival at the execution queue to its successful completion is used as the main performance measure. Similarly, this time is the main performance measure in our study here.

A detailed performance study of distributed and replicated databases, based on analytical methods, is presented in [24]. A set of alternatives for modelling network communication, data replication and query processing is proposed, and the various interdependencies between the components of the model are discussed. The distributed data model with random data replication resembles the distributed P2P environment we are concerned with in this paper.

For some applications, content-based publish/subscribe is an alternative technology to ECA rules. P2P networks that support publish/subscribe, such as [11, 33] for example, support more sophisticated distributed event definition and detection than ECA rules. On the other hand, ECA rules allow the definition and execution of more complex actions than just simple notifications.

The architecture of RDFTL [14] is an example of a schema-based P2P system, similar in particular to Edutella [23]. In [23] and [14] the metadata distribution allows hybrid fragmentation, with possible replication between peers. The SQPeer [20] P2P architecture similarly does not impose any particular data fragmentation or replication policy; however, unlike RDFTL, it allows a peer to belong to more than one peergroup. Piazza [32] provides semantic integration and global querying of heterogenous data distributed over a P2P network, where each peer supports its own schema. The XP2P system [9] addresses the problem of distributed XPath query processing in a P2P network based on distributed hash tables. XP2P provides XML data handling in a structured P2P network along with algorithms that facilitate locating the peers that are able to evaluate a given XPath expression. A description of replication strategies that improve the performance of requests in unstructured P2P networks is presented in [12], along with a set of experimental results regarding various performance factors.

3 Background

3.1 The RDFTL Language

In this section we give an overview of RDFTL to the level of detail necessary for Sections 4 onwards of this paper. RDFTL operates over RDF graphs [35, 37]. It is assumed that these RDF graphs conform to one or more RDFS schemas [36], in the sense that (a) every resource in the RDF graph belongs to an RDFS class (in addition to belonging to the default `rdfs:Resource` class); (b) every property in the RDF graph is declared in the RDFS schema,

along with domain and range constraints; (c) the subject and object of every property in the RDF graph are of the declared subject and object type of the property in the RDFS schema.

RDFTL uses a path-based query sublanguage, syntactically similar to XPath [34], for defining queries over an RDF graph. The syntax of this sublanguage is as follows, where e is a query, p is a path expression, q is a qualifier, *uri* is a URI, *arc_name* is a predicate and s is a string:

$$
\begin{aligned}
e &::= \text{"}resource(\text{"}uri\text{"})\text{"}\ (\text{"}/\text{"}p)? \\
p &::= p\text{"}/\text{"}p \mid p\text{"}[\text{"}q\text{"}]\text{"} \mid \text{"}target(\text{"}arc\_name\text{"})\text{"} \\
&\quad \mid \text{"}source(\text{"}arc\_name\text{"})\text{"} \\
q &::= q\ \text{"}and\text{"}q \mid q\ \text{"}or\text{"}\ q \mid \text{"}not\text{"}\ q \mid p \mid p\ \text{"}=\text{"}\ s \mid p\text{"} \neq \text{"}\ s
\end{aligned}
$$

Each RDFTL rule has an optional preamble consisting of one or more namespace definition clauses and a set of *let-expressions* of the form `let` *variable* := e associating a variable with a query e.

The event part of an RDFTL rule describes updates whose occurrence will cause the rule to trigger, and is an expression of one of the following three forms:

1. `(INSERT | DELETE)` e `[AS INSTANCE OF` *class*`]`
2. `(INSERT | DELETE)` *triple*
3. `UPDATE` *upd_triple*

Form 1 detects insertions or deletions of resources specified by the expression e. e is a query, which evaluates to a set of nodes. Optionally, *class* is the name of the RDFS schema class to which at least one of the nodes identified by e must belong in order for the rule to trigger. The rule is triggered if the set of nodes returned by e includes any new node (in the case of an insertion) or any deleted node (in the case of a deletion) that is an instance of *class*, if specified. The system-defined variable `$delta` is available for use within the condition and actions parts of the rule, and its set of instantiations is the set of new or deleted nodes that have triggered the rule.

Form 2 detects insertions or deletions of arcs specified by *triple*, which has the form *(source_node, arc_name, target_node)* where *source_node* and *target_node* may be expressions of the form e or variables defined in the rule's preamble. The wildcard '`_`' is allowed in the place of any of a triple's components. The rule is triggered if an arc labelled *arc_name* from *source_node* to *target_node* is inserted/deleted. The variable `$delta` has as its set of instantiations the triples which have triggered the rule; the individual components of these triples are identified by `$delta.source`, `$delta.arc_name` or `$delta.target`.

Form 3 similarly detects updates to the target nodes of arcs, specified by *upd_triple* which has the form *(source_node, arc_name, old_target_node* $\rightarrow$ *new_target_node)*. The wildcard '`_`' is allowed in the place of any of these components. The rule is triggered if an arc labelled *arc_name* from *source_node* changes its target from *old_target_node* to *new_target_node*. The variable `$delta` has as its set of instantiations the triples which have triggered the rule.

The condition part of rule is a boolean-valued expression which may consist of conjunctions, disjunctions and negations of queries.

The actions part of a rule is a sequence of one or more actions. Actions can `INSERT` or `DELETE` a resource — specified by its URI — and `INSERT`, `DELETE` or `UPDATE` an arc. The

actions language has the following form for each one of these cases, where triples in the actions part have a similar form as in the event part:

1. `INSERT` *e* `AS INSTANCE OF` *class*

   `DELETE` *e* `[AS INSTANCE OF` *class*`]`

   for expressing insertion or deletion of a resource, where the `AS INSTANCE OF` keyword classifies the resource to be deleted or inserted.

2. `(INSERT | DELETE)` *triple* `(','` *triple*`)*`

   for expressing insertion or deletion of the arcs(s) specified.

3. `UPDATE` *upd_triple* `(','` *upd_triple*`)*`

   for updating arc(s) by changing their target node.

The condition and action parts of an RDFTL rule may contain occurrences of the `$delta` variable in place of a named resource in a query, or a component of a triple. If neither the condition nor the action part contain occurrences of `$delta`, then the rule is a *set-oriented rule*, otherwise it is an *instance-oriented rule*.

A set-oriented rule *fires* if it is triggered and its condition evaluates to true. A copy of the rule's action part is executed as a new transaction. An instance-oriented rule fires if it is triggered and its condition evaluates to true for some instantiation of `$delta`. A copy of the rule's action part is executed as a new transaction for each value of `$delta` for which the rule's condition evaluates to true, in each case substituting all occurrences of `$delta` within the action part by one specific instantiation for `$delta`.

In general, many rules may fire as a result of an event occurrence. The set of rules is partially ordered by a *rule precedence* relationship, *prec*, which is specified by the rule designer or application. If two rules r_i and r_j fire and r_i *prec* r_j then the updates generated by r_i are executed before those generated by r_j. If r_i and r_j are not related by *prec*, then it is assumed that they *commute*, i.e. that the same final RDF graph will result when rule execution terminates irrespective of the order of scheduling of r_i and r_j. Similarly, it is assumed that the order of execution of instances of the action part of an instance-oriented rule is arbitrary, and that the same final RDF graph will result when rule execution terminates irrespective of this order. [14] discusses conservative tests for determining the termination and confluence of sets of RDFTL rules.

3.2 RDFTL Rules in P2P Environments

[14] describes a system for processing RDFTL rules in P2P environments. The rule processing functionality in this system is provided by a set of services that constitute the *RDFTL ECA Engine*. This set of services acts as a wrapper over a distributed set of RDF/S repositories, exploiting their query, storage and update functionality.

The architecture of the system is illustrated in Figure 1. Each superpeer may be supervising a group of further peers, termed its *peergroup*. At each superpeer there is one ECA Engine installed. Each peer or superpeer hosts a fragment of an overall global RDFS schema. The metadata distribution allows hybrid fragmentation, with possible replication between peers. The fragment of the global RDFS schema stored at a peer may change as a result of changes

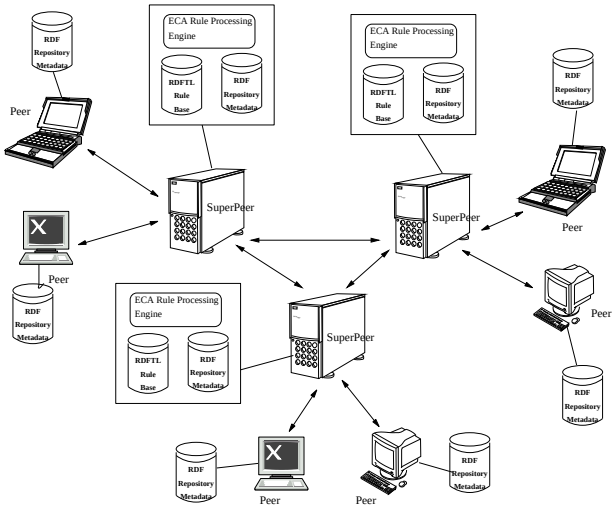

Figure 1: P2P System Architecture

in the peer's RDF/S descriptions. Peers notify their supervising superpeer of any updates to their local RDF/S repository. Peers may dynamically join or leave the network at any time.

Each superpeer's RDFS schema is a superset of its peergroup's individual RDFS schemas. Each superpeer defines access privileges over the classes and properties in its RDFS schema (read-only, read-write, private) describing the corresponding access level to the instances of each class and property. More fine-grained access privileges are also allowed on specific RDF resources and triples. These facilities allow a superpeer to specify which information can be shared with other superpeers outside its peergroup.

An ECA rule generated at one site of the network might be replicated, triggered, evaluated, and executed at different sites. Within the event, condition and action parts of ECA rules there might be references to specific RDF resources.

Whenever a new ECA rule r is generated at a peer P, it is sent to P's superpeer for syntax validation, translation into the local repository's query and update language, and storage. From there, r will also be forwarded to all other superpeers, and a replica of it will be stored at those superpeers where an event may occur that may trigger r's event part, i.e. those superpeers that are *e-relevant* to r (see below). A rule r has a globally unique identifier of the form $SP_i.j$, where SP_i is the originating superpeer identifier and j a locally unique identifier for the rule in SP_i's rule base.

At run-time rules are triggered by events occurring within a single peer's local RDF repository, i.e. there is no distributed event detection. Also, each particular copy of a rule's action part executes within a single peer's RDF repository, i.e. there is no distributed update execution. If there is a need to distribute a sequence of updates across a number of peers in reaction to some event, then rather than specifying one rule of the form `on` e `if` c `do` $a_1, \ldots, a_n$ instead, n rules $r_1, \ldots, r_n$ can be specified, where each r_i is `on` e `if` c `do` a_i and r_1 *prec* r_2 *prec* ... *prec* r_n.

There are three types of *relevance* of a rule r to an RDF schema S:

- r is *e-relevant* to S if each of the queries that either appear in the event part of r or are used by the event part through variable references, can be evaluated on S, i.e., each step in each path expression exists in S.

- r is *c-relevant* to S if some step in one of the queries referenced by the condition part of r can be evaluated on S (unlike events and actions, conditions may be evaluated at multiple sites).

- r is *a-relevant* to S if all actions in the action part of r are *a-relevant* to S. An individual action is a-relevant to S if it satisfies one of the following: (i) If it is a deletion or insertion of resources that uses `AS INSTANCE OF` *`class`*, then *`class`* must be in S. (ii) If it is a deletion of resources that does not use `AS INSTANCE OF` *`class`*, then the most specific class of resources that the path expression in the deletion would return must be in S. (iii) If it is an action over triples that uses a property p, then p must be in S. If it is a deletion of triples that uses the wildcard '_' instead of a property (the only action allowed to do this), then the classes of resources returned by the path expressions involved in the deletion must exist in S.

A peer or superpeer is e-relevant, c-relevant or a-relevant to a rule r if r is e-, c- or a-relevant, respectively, to the peer or superpeer's RDFS schema.

At each superpeer, each rule is annotated with the IDs of local peers that are e-relevant, c-relevant and a-relevant to it. These annotations are kept synchronised with changes in peers' and superpeers' schemas.

3.3 P2P Rule Execution

The RDF graph is fragmented, and possibly replicated, amongst the peers, and each superpeer manages its own local rule execution schedule. Each execution schedule at a superpeer is a sequence of updates which are to be executed on the fragment of the global RDF graph which is stored at the superpeer or its local peergroup. Each superpeer coordinates the execution of global transactions that are initiated by that superpeer, or by any peer in its local peergroup.

Whenever an update u is executed at a peer P, P notifies its supervising superpeer SP. SP determines whether u may trigger any ECA rule in its rule base whose event part is annotated with P's ID. If a rule r may have been triggered, then SP will send r's event query to P to evaluate.

If r has indeed been triggered, its condition will need to be evaluated, after generating an instantiation of it for each value of the `$delta` variable if this is present in the condition.

If a condition evaluates to true, SP will send each instance of r's action part (one instance if r is a set-oriented rule, and one or more instances if r is an instance-oriented rule) to the local peers that are a-relevant to it. All instances of r's actions part will also be sent to all other superpeers of the network. All superpeers that are a-relevant to r will consult their schemas and access privileges in order to determine whether the updates they have received can be scheduled and executed on their local peergroup.

In summary therefore, local execution of the update at the head of a local schedule may cause events to occur. These events may cause rules to fire, modifying the local schedule or remote schedules with new updates to be executed.

4 Analytical Performance Model

We have chosen *update response time* as the main performance criterion of the system described above, where we define this to be the mean time taken to complete all rule execution

resulting from a single update submitted by a top-level transaction. Transactions consisting of queries and updates are submitted by applications to peers or superpeers. Updates may cause rules to fire, which may in turn cause the firing of further rules, increasing the network traffic as well as the load on peers and superpeers. Queries submitted by a top-level transaction cannot cause rule firing and can only increase the load on peers and superpeers. In designing the analytical model, we have made some assumptions as detailed below.

4.1 Homogeneity Assumption

In common with other performance studies (e.g. [24]), we make a homogeneity assumption separately for the peers and superpeers of the network. For both these two sets of servers, this assumption asserts identical workloads, the same service capacity and the same amount of data per site. For the superpeers, it also implies the same number of rules per rule base.

For the network communication, the homogeneity assumption implies symmetrical communication between superpeers, and between peers and superpeers i.e. the average number of messages from a site A to a site B equals the average number of messages from site B to site A. The size of the messages exchanged is considered to be fixed.

We also assume a balanced peer distribution amongst superpeers, with each peergroup having the same number of peers.

4.2 Rule triggering assumptions

In our experience, the level of ECA rule triggering in database applications tends to be shallow, in that the probability of having k levels of triggering in response to some top-level update decreases substantially as k increases. We regard level 0 as the top-level update, and this may cause some rules to fire. Updates occurring at level $i+1$ result from rules that have been fired by updates occurring at level i.

$p_{fire}(i)$ denotes the probability that an update occurring at level i of rule execution causes a given ECA rule to fire. This probability depends on the probability $p_{mt}(i)$ that a given ECA rule may be triggered by an update[1], the probability p_t that a rule that may be triggered is actually triggered, and the probability p_f that a rule that has been triggered actually fires. $p_{fire}(i)$ is given by the following equation:

$$p_{fire}(i) = p_{mt}(i) \cdot p_t \cdot p_f \tag{1}$$

We assume that the "may be triggered" probability $p_{mt}(i)$ follows a geometric distribution with a constant reduction factor of p_{reduct} at each level, while p_t and p_f remain constant regardless of the triggering level. So $p_{mt}(i)$ is given by the following equation:

$$p_{mt}(i) = p_{mt}(0) \cdot p_{reduct}^{i} \tag{2}$$

where $p_{mt}(0)$ denotes the probability that a rule may be triggered as a result of a top-level update.

As a consequence of equations 1 and 2:

$$p_{fire}(i) = p_{mt}(0) \cdot p_t \cdot p_f \cdot p_{reduct}^{i} \tag{3}$$

[1]A rule "may be triggered" by an update if its event part matches syntactically this update. The rule will actually be triggered only if the set of instantiations of the `$delta` variable intersects the data returned by evaluating the event part of the rule on the post-update state.

implying that p_{fire} also reduces geometrically with the level i. At each level i, the number of rules $r_{fire}(i)$ that fire for each event that occurs at level i is given by

$$\begin{aligned} r_{fire}(i) &= p_{fire}(i) \cdot n_{rules} \\ &= p_{mt}(i) \cdot p_t \cdot p_f \cdot n_{rules} \end{aligned} \tag{4}$$

Similarly, the number of rules that may be triggered by an update at level i is given by

$$r_{mt}(i) = p_{mt}(i) \cdot n_{rules} \tag{5}$$

4.3 System Modelling

In our model we assume two kinds of queues. A *transaction queue* at each peer that accepts queries or updates arising from rule execution, and an *action scheduler queue* at each superpeer that queues transactions resulting from rule firing which are then dispatched for execution to the appropriate transaction queues of peers in the peergroup. For each queue we assume a FCFS (First Come First Served) service discipline. New queries/updates that arrive at a peer are placed at the tail of the peer's transaction queue for execution, and the same happens at the action scheduler's queue in superpeers.

We assume that the traffic in transaction queues at peers is low enough so that each time a query/update arrives for execution at a peer, it immediately receives service without having to spend time waiting in the queue. The query/update arrival rate is modelled as a Poisson process, which means that the arrival of a new item does not depend on any previous item (the process is memoryless) and the inter-arrival time is exponentially distributed. The exponential distribution of inter-arrival time leads to the *service time* also following an exponential distribution. By service time we mean the time for a query to be evaluated or an update to be executed in a peer's repository. An empirical justification of the service time's exponential distribution is given in [24] and is as follows: The time required for a query/update depends on the number of data items accessed. Queries/updates that access a small number of data items are more frequent than those accessing a large number of data items. This leads to a geometrically distributed number of data items accessed per query/update. So the service time can be assumed exponential, which is the continuous version of geometrical.

Using Kendall's notation [17], the transaction queues and the action scheduler queues are queues of the form $M/M/1$, where M indicates exponential distribution for both the process arrival rate and the service time, and 1 specifies that each peer or superpeer provides one single service point.

Regarding the network communication within the system, we assume that the communication channel between superpeers, and between superpeers and peers, can be modelled as a server with an infinite number of service points, introducing a delay to all messages passed depending on their size and the network bandwidth. The queue model description of this is $M/D/\infty$, where D indicates deterministic service time and ∞ an unlimited capacity network[2].

We assume that the size of messages is fixed at $size_m$ (in bytes), and that the communication bandwidth between superpeers is twenty times greater than the bandwidth between

[2]A more accurate but more complex way to model the inter-superpeer communication and the communication between superpeers and peers would be to assume that each communication channel is a queuing network of N $M/M/1$ queues. So we could define that for the inter-superpeer communication case there are N_{sp} channels and for the superpeer-peer communication case N_p channels.

peers and superpeers. So the network communication delay introduced for messages between a peer and a superpeer, t_{delay}^{P-SP}, is given by the following (where multiplying by 8 converts the message size $size_m$ from bytes to bits and bps represents the network communication bandwidth in bits per second between a peer and superpeer):

$$t_{delay}^{P-SP} = \frac{8 \cdot size_m}{bps} \tag{6}$$

and for messages transmitted between two superpeers the delay, t_{delay}^{SP-SP}, is given by:

$$t_{delay}^{SP-SP} = \frac{t_{delay}^{P-SP}}{20} \tag{7}$$

4.4 Modelling Update Response Time

We recall that the update response time is the mean time taken to complete all rule execution resulting from a single update submitted to a peer or a superpeer by a top-level transaction. This update response time, $\overline{R}_{update}$, can be decomposed as follows:

$$\overline{R}_{update} = \overline{R}_{event} + \overline{R}_{cond} + \overline{R}_{action} \tag{8}$$

where $\overline{R}_{event}$ is the mean time taken for all rule event processing, $\overline{R}_{cond}$ the mean time taken for all rule condition processing and $\overline{R}_{action}$ the mean time taken for all rule action processing during the rule execution following a top-level update. We now consider each of these three components in turn.

4.4.1 Event Response Time ($\overline{R}_{event}$)

For each level of triggering i, the mean time taken to process the event part of a single rule at level i, denoted by $\overline{R}_{event}^{rule}(i)$, is the sum of the mean time $\overline{T}_{event}^{db}(i)$ spent in peer database processing at level i plus the mean time $\overline{T}_{event}^{net}(i)$ spent in network transmission at level i for the rule:

$$\overline{R}_{event}^{rule}(i) = \overline{T}_{event}^{db}(i) + \overline{T}_{event}^{net}(i) \tag{9}$$

When an event occurs at a peer P, details such as the type of the event, the data items that were affected (i.e. the sets of resources inserted or deleted, or the sets of triples that have been inserted, deleted or updated), and the time the event occurred, are wrapped into a message and transmitted to the coordinating superpeer SP. This determines the set of rules in its rule base that may be triggered by the event. The event part of each of the $r_{mt}(i)$ rules that may be triggered is sent back to P for evaluation. The evaluation results for each event part are transmitted back to SP where they are matched against the set of data items affected by the event. If the intersection of the two sets is non-empty, then the rule is actually triggered.

These processing steps involve contacting the network services three times, the repository services at peer P $r_{mt}(i)$ times, plus matching of the affected data items against the evaluation results of the event part of each of the $r_{mt}(i)$ rules. The network processing time needed for

the event part is therefore

$$\begin{aligned}\overline{T}^{net}_{event}(i) &= t^{P-SP}_{delay} \cdot \frac{m-1}{m} \\ &+ 2 \cdot t^{P-SP}_{delay} \cdot \frac{m-1}{m} \cdot p_{mt}(i) \cdot n_{rules} \end{aligned} \tag{10}$$

where t^{P-SP}_{delay} is the constant network delay caused by each message sent, m is the number of peers in a peergroup (including the superpeer itself), and $p_{mt}(i)$ is the probability that a rule may be triggered at level i. The factor $\frac{m-1}{m}$ represents the probability the event has not occurred on the superpeer, assuming events are equally likely to occur at any peer in the peergroup. If the event does occur on the superpeer, then no network transmission will be necessary.

For the processing at peers' and superpeers' repositories, the total time consumed at level i is:

$$\overline{T}^{db}_{event}(i) = r_{mt}(i) \cdot t_q + r_{mt}(i) \cdot t_q = 2 \cdot n_{rules} \cdot p_{mt}(i) \cdot t_q \tag{11}$$

Here, t_q is the time needed for a rule's event query to be evaluated on a peer's repository, and there is one event query to be evaluated for each of the $r_{mt}(i)$ rules that may be triggered. We have used the same quantity, t_q, for the time needed at the superpeer for evaluating the intersection of the results of an event query with the set of data items affected by the event, and again there are $r_{mt}(i)$ such intersections to be evaluated. We also make use of the assumption that the transaction traffic at peers is low enough so that each time a new query arrives at a peer it finds the transaction queue empty and is executed immediately.

Assuming that the action part of each rule contributes N_{action} updates to an action schedule (see below), then substituting equations 10 and 11 into equation 9, and summing over all k triggering levels, we obtain the mean time taken to process all event queries over all k triggering levels during the rule execution following a top-level update as:

$$\begin{aligned}\overline{R}_{event} &= \sum_{i=1}^{k} \{ r_{fire}(i-1) \cdot N_{action} \cdot [t^{P-SP}_{delay} \cdot \frac{m-1}{m} \\ &\quad + 2 \cdot n_{rules} \cdot p_{mt}(i) \cdot (t^{P-SP}_{delay} \cdot \frac{m-1}{m} + t_q)] \}\end{aligned}$$

where the factor $r_{fire}(i-1) \cdot N_{action}$ represents the total number of individual updates caused by the previous level of triggering.

4.4.2 Condition Response Time ($\overline{R}_{cond}$)

From the set of may-be-triggered rules, only a subset may actually be triggered. The number of rules $r_t(i)$ that are actually triggered at level i is given by $r_{mt}(i) \cdot p_t$, where p_t is the probability that a may-be-triggered rule is actually triggered. For each of these $r_t(i)$ rules, the time needed for its condition part to be evaluated, is given by the following equation:

$$\overline{R}^{rule}_{cond}(i) = \overline{T}^{db}_{cond}(i) + \overline{T}^{net}_{cond}(i) \tag{12}$$

where $\overline{T}^{db}_{cond}(i)$ is the time spent on database processing and $\overline{T}^{net}_{cond}(i)$ the time spent on network transmission.

The condition part of each of the rules triggered needs to be evaluated in a distributed manner, in order to determine the subset of rules that will finally fire. This involves generating the appropriate number of instances of the condition part (one or more in the case of an instance-oriented rule, one in the case of a set-oriented rule), determining to which of the peers in the peergroup subqueries of the condition should be dispatched, transmission of the sub-queries to the appropriate peers, evaluation of the subquery at each peer, and finally transmission of the evaluation results back to the superpeer.

We note here that the system described in [14] assumes that rule conditions are evaluated locally within a peergroup. Global rule condition evaluation across multiple superpeers would need to use global P2P query processing techniques, which would generally increase the update response time of ECA rule processing. Its effect on the observed trends of our experimental results presented in the next section would clearly depend on the complexity and scalability of the query processing algorithms employed. For example, [29] discusses query routing in P2P networks that use the HyperCup topology and shows that schema-based clustering of peers at the superpeers improves the network performance significantly. In order not to confuse the performance of global P2P query processing with the additional ECA rule processing functionality that is the focus of this paper, we retain in this paper the assumption that rule conditions are evaluated locally within a peergroup.

We assume that each condition part generates an average N_{cond} of instances. Each of these N_{cond} instances has an average number of n_{steps} steps within its constituent queries. The probability $p_{annot_{cond}}$ that a condition instance is relevant to one of the m peers of the peergroup is expressed by the probability that at least one of the steps within its constituent queries can be evaluated at the peer. Each query "step" is either a specific resource URI or the name of a property. Indexes are maintained at each superpeer which state which resources are present in the RDF repository of each peer of its peergroup, and also which RDFS properties appear within triples in each of the peers' RDF repositories (see [14] for details of these indexes; we assume here that the cost of index look-up is negligible compared with the other system costs). Thus, $p_{annot_{cond}}$ is given by:

$$p_{annot_{cond}} = 1 - (1 - p_s)^{n_{steps}}$$

where p_s is the probability that a query step can be evaluated at the given peer.

We note that the probability p_s also captures the amount of schema and metadata replication between peers in the overall network. A higher p_s value implies greater replication and a lower implies less replication. A value of $p_s = 1$ corresponds to full schema and metadata replication at all peers in the network.

From the above, the mean time needed for a rule's condition part to be evaluated is:

$$\overline{T}^{db}_{cond}(i) = t_q \cdot N_{cond} \cdot n_{steps} \cdot p_{annot_{cond}} \cdot m \tag{13}$$

where we assume that the time taken to evaluate one step of a query within a rule's condition part is t_q.

The mean time spent on network transmission for the evaluation of a rule's condition part is given by

$$\begin{aligned} \overline{T}^{net}_{cond}(i) &= 2 \cdot t^{P-SP}_{delay} \cdot N_{cond} \cdot n_{steps} \\ &\quad \cdot p_{annot_{cond}} \cdot m \cdot \frac{m-1}{m} \end{aligned} \tag{14}$$

with the factor $\frac{m-1}{m}$ representing, once again, the probability that no transmission is necessary.

From equations 12, 13 and 14 we obtain the the mean time needed for the evaluation of the condition part of a single rule as:

$$\overline{R}^{rule}_{cond}(i) = N_{cond} \cdot n_{steps} \cdot p_{annot_{cond}} \cdot [t_q \cdot m + 2 \cdot t^{P-SP}_{delay} \cdot (m-1)] \tag{15}$$

So, the mean time taken to process all condition queries over all k triggering levels during the rule execution following a top-level update is as follows, recalling that $r_t(i)$ rules are triggered at the ith triggering level:

$$\begin{aligned} \overline{R}_{cond} &= \textstyle\sum_{i=0}^{k} \left(r_t(i) \cdot \overline{R}^{rule}_{cond}(i) \right) \\ &= p_t \cdot n_{rules} \cdot N_{cond} \cdot n_{steps} \cdot p_{annot_{cond}} \\ &\quad \cdot [t_q \cdot m + 2 \cdot t^{P-SP}_{delay} \cdot (m-1)] \cdot \textstyle\sum_{i=0}^{k} p_{mt}(i) \end{aligned}$$

4.4.3 Action Response Time ($\overline{R}_{action}$)

The number $r_{fire}(i)$ of the rules that fire at the ith level of triggering, is expressed by the product of the probability p_f of a triggered rule to fire and the number $r_t(i)$ of rules that were triggered at a particular triggering level i, $r_{fire}(i) = r_t(i) \cdot p_f$. The mean time needed for the execution of the action part of each of the rules that fire is again expressed as the sum of the time spent on network transmission and the time consumed on path expression evaluation and update execution at the peers' RDF repositories:

$$\overline{R}^{rule}_{action}(i) = \overline{T}^{db}_{action}(i) + \overline{T}^{net}_{action}(i) \tag{16}$$

Each instance of the action part of each rule that fires (one instance in the case of a set-oriented rule, one or more instances in the case of instance-oriented rules) is sent to the peers of the local peergroup according to the annotations on the rule's action part and consulting also the superpeer resource indexes for the presence of specific URIs at peers. Each instance of the action part of each rule is also sent to all other superpeers of the network. Depending on the data access rights and on the schema and resources present at each remote superpeer, the rule may scheduled and executed there.

The time spent in network transmission, according to the above process, is given by the following:

$$\begin{aligned} \overline{T}^{net}_{action}(i) &= [t^{SP-SP}_{delay} \cdot p_{annot_{action}} \cdot m \cdot \frac{m-1}{m} \\ &+ (t^{SP-SP}_{delay} \cdot n_{hops} + t^{SP-SP}_{delay} \cdot (n-1) \cdot p_{allow} \cdot \\ &\quad p_{annot_{action}} \cdot m \cdot \frac{m-1}{m})] \cdot N_{action} \end{aligned} \tag{17}$$

Here n_{hops} expresses the number of hops, i.e. message transmission steps, required for an instance of a rule's action part to be transmitted to all of the superpeers of the network. This number depends on the network topology and the routing strategy followed. At each of the $n-1$ remote superpeers, the instance of the action part will be scheduled for execution depending on the probability p_{allow} that the superpeer has within its peergroup schema and metadata the resources necessary to execute the instance and that its data access privileges allow access to these. If the execution is possible, the updates comprising the instance of the action part will be sent to $p_{annot_{action}} \cdot m \cdot \frac{m-1}{m}$ peers of the peergroup. Here $p_{annot_{action}}$

expresses the probability that all the steps within the path expressions of the instance of the action part can be executed at a given peer, and is given by

$$p_{annot_{action}} = p_s^{n_{steps}} \tag{18}$$

where, as with rule conditions, p_s expresses the probability that a step can be evaluated at the peer and we assume the same number of steps, n_{steps}, in a rule's action part as in its condition part.

As mentioned earlier, instances of rules' action parts are placed on superpeers' action scheduling queues. We assume that there are an average of N_{action} updates placed by a rule and that each rule's action part contains on average $size_{ap}$ individual actions. So the average number of instances ℓ_{inst} an action part contributes is:

$$\ell_{inst} = \frac{N_{action}}{size_{ap}}$$

After an instance of the action part is placed on an action scheduling queue at level i, each update within it waits for a time of $\overline{W}(i)$ before being dispatched, as part of the whole instance, to the appropriate peers of the local peergroup, where it receives a service time of t_{srv}. Thus, the mean time spent on the execution of an update within an instance of an action part within a peergroup is:

$$T_{update\_exec}(i) \quad = \quad t_{srv} + \overline{W}(i) \tag{19}$$

where t_{srv} is given by

$$t_{srv} \quad = \quad p_{annot_{action}} \cdot m \cdot t_q$$

since an update will be executed, as part of the instance, on $p_{annot_{action}} \cdot m$ peers of the peergroup and we assume that the time needed for the execution of an update is t_q.

The mean waiting time in the $M/M/1$ action scheduling queue at level i of triggering is given by the following equation [17]:

$$\overline{W}(i) = \frac{\lambda_{total}(i) \cdot t_{srv}^2}{1 - \lambda_{total}(i) \cdot t_{srv}} \tag{20}$$

Here, $\lambda_{total}(i)$ is the total arrival rate of updates, in total, at a peergroup at level i of triggering, given by

$$\lambda_{total}(i) \quad = \quad \lambda_{ext} + \lambda_{int}(i) \tag{21}$$

where λ_{ext} is the arrival rate of "top-level" updates that are submitted from outside the rule processing system, and $\lambda_{int}(i)$ is the arrival rate of updates that are generated as a result of rule firing at level i.

Each of the externally arriving top-level updates will cause a first level of triggering. Each of the updates contained within the transactions generated as a result of this rule triggering may cause further rules to fire, causing more transaction traffic to be generated and so on.

So the total arrival rate of updates at a peergroup at level i of triggering can be expressed as:

$$\begin{aligned}\lambda_{total}(i) &= \lambda_{ext} + \lambda_{ext} \cdot r_{fire}(1) \cdot N_{action} \\ &+ (\lambda_{ext} \cdot r_{fire}(1)) \cdot r_{fire}(2) \cdot N_{action}^2 \\ &+ \ldots \\ &+ (\lambda_{ext} \cdot r_{fire}(1) \ldots \cdot r_{fire}(i-1)) \cdot r_{fire}(i) \cdot N_{action}^i \\ &= \lambda_{ext} \cdot [1 + \sum_{a=1}^{i} N_{action}^a \cdot \prod_{b=1}^{a} r_{fire}(b)] \end{aligned} \quad (22)$$

where $r_{fire}(i)$ is the number of rules that fire at triggering level i, and $N_{action} = \ell_{inst} \cdot size_{ap}$ is the total number of individual updates each of the $r_{fire}(i)$ rules generates.

According to equations 1, 3, 4 and 22, the transaction arrival rate at level i is

$$\begin{aligned}\lambda_{total}(i) &= \lambda_{ext} \cdot (1 + \\ &\sum_{a=1}^{i} n_{rules}^a \cdot p_{fire}(0)^a \cdot N_{action}^a \cdot \prod_{b=1}^{a} p_{reduct}^b)\end{aligned}$$

The time for all the updates within instances of the action part at level i to be executed is $T_{action\_exec}(i) = N_{action} \cdot T_{update\_exec}(i)$.

The time needed for all the ℓ_{inst} instances of a rule's action part to be executed with probability p_{allow} on the rest of the $n-1$ remote peergroups is also $T_{action\_exec}(i)$, and so the total time needed for the execution of the action part of a rule at level i is:

$$\overline{T}_{action}^{db}(i) = T_{action\_exec}(i) \cdot [1 + (n-1) \cdot p_{allow}] \quad (23)$$

Using equation 16, the fact that $r_{fire}(i)$ rules fire at level i, and that there are k triggering levels, we obtain the time taken for all rule action processing as

$$\begin{aligned}\overline{R}_{action} &= \sum_{i=1}^{k} r_{fire}(i) \cdot \left(\overline{R}_{action}^{rule}(i)\right) \\ &= \sum_{i=1}^{k} r_{fire}(i) \cdot (\overline{T}_{action}^{db}(i) + \overline{T}_{action}^{net}(i)) \end{aligned} \quad (24)$$

into which the right-hand sides of equations 17 and 23 can be substituted.

5 Experimental Results

As well as developing the analytical performance model described in the previous section, we have also developed a simulator of the RDFTL system in order to increase the accuracy of the performance study and validate the predictions of the analytical model. We have conducted experiments with the analytical model and with the simulator for two different network topologies, random flooding and HyperCup [28]. We have examined the update response time on each, varying the n, n_{rules} and p_s parameters. We note here that the network topology relates to the communication between superpeers, as each (non-superpeer) peer has only one active network connection with its supervising superpeer only. The results

of our experiments are discussed in Section 5.1 for the analytical model and Section 5.2 for the simulator.

For the purposes of these experiments, we have used our RDFTL system implementation in order to take measurements for the system parameters λ_{ext}, $size_m$, bps and t_q, and the values obtained for these are shown in Table 1. In the absence of information about real RDFTL rule sets for real applications, we have fixed the values of the ECA-rule related parameters k, N_{cond}, N_{action}, $size_{ap}$, p_{reduct}, p_{allow}, p_t, p_f, $p_{mt}(0)$ and n_{steps} as shown in Table 1. For the e-learning and e-science applications currently envisaged for RDFTL, we believe that these values are representative of the likely rule sets that will arise. The value of N_{action} is relatively low as RDFTL rules operate on metadata and so "bulk" updates are likely to be infrequent. The value of k is of the same order of magnitude as adopted in commercial active DBMS as an upper limit for the number of recursive rule firings allowed. The number of peers in each peergroup is fixed at $m = 20$.

In our performance study below, we consider the shape and the general trends of the performance graphs as the primary result rather than the absolute values obtained. We will see that varying the values of the n, n_{rules} and p_s parameters affects the absolute performance values but not their general trend. Similar sets of experiments with different settings of the fixed parameters have also been conducted and this affects only the absolute performance values, and not the performance trends.

5.1 Analytical Study Results

In our analysis, we examine for each of the two network topologies the update response time with respect to varying numbers of peergroups in the network (n), varying numbers of rules per rule base (n_{rules}) and varying data replication (p_s).

Parameter	**Base setting**
λ_{ext}	20 TPS
$size_m$	10 kbytes
bps	512 kbps
t_q	1 sec
k	30
N_{cond}	4
N_{action}	8
$size_{ap}$	4
p_{reduct}	0.2
p_{allow}	0.9
p_t	0.5
p_f	0.5
$p_{mt}(0)$	0.1
n_{steps}	4
m	20

Table 1: Parameter Base Values

In the first topology, the superpeers are connected at random and any message between them is broadcast from the originating superpeer to all its neighbouring superpeers, and from

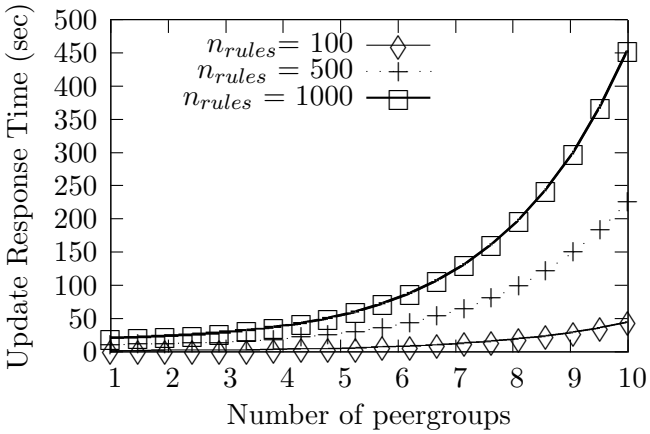

Figure 2: Model, Replication 10%, Full Net

there to all their neighbours, and so forth, flooding the network until the message reaches all the superpeers. This simple topology does not place any guaranteed upper bound on the number of hops that will be necessary for a message to reach all superpeers. It also does not prevent a message being received more than once by the same superpeer.

Figure 2 shows how the update response time varies with this random topology as the number of peergroups n increases, with the data replication p_s being set to 10% and the number of rules per rule base n_{rules} set to 100, 500 and 1000. From the shape of these curves it is clear that the system does not scale well.

As the number of peergroups increases, the update response time rapidly rises towards very high values. For a relatively small number of rules per rule base (100) and for up to 10 peergroups we obtain acceptable update response times, due to the relatively low number of available communication paths between superpeers. As the number of peergroups n increases, and with that the number of possible communication paths between superpeers, the rapidly rising values of the update response time make the system unstable and unusable. Similar behaviour is observed with larger values of n_{rules} except that the system becomes unstable at lower values of n. Similar sets of experiments conducted for higher values of p_s result in graphs with similar upwards trends, except that the absolute values of the update response time are larger and the system becomes unstable at lower values of n. This is to be expected as the presence of data in more peergroups increases the number of peers that a rule which has fired can be executed on, thus increasing the network traffic and repository load, and the number of further recursive rule firings.

The HyperCup topology [28] guarantees that: each peer receives a message only once; if the number of peers is N, a total of $N-1$ hops are required to reach all peers; and the most distant peers are reached after $\log_2 N$ hops (assuming a hypercube topology, which is what we have adopted in our experiments).

Figure 3 shows how the update response time varies with the HyperCup topology as the number of peergroups n increases, with the data replication p_s being set to 10% and the number of rules per rule base, n_{rules}, set to 100, 500 and 1000. We see that the system now shows good scalability, and the update response time increases linearly with n.

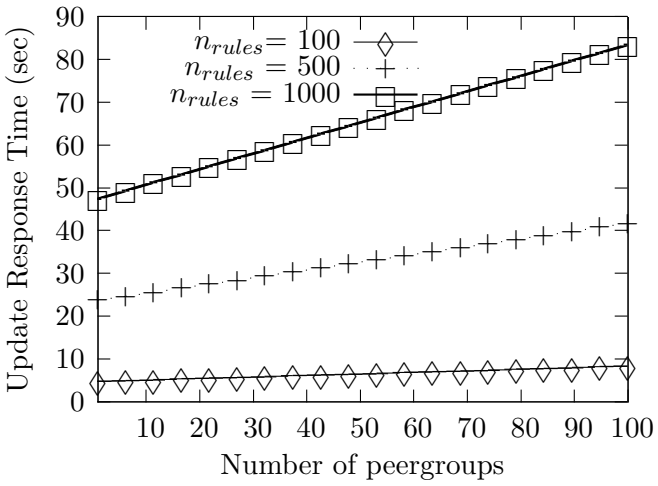

Figure 3: Model, Replication 10%, HyperCup

Compared with the random topology, the system remains stable and the update response time within reasonable boundaries even for large networks and numbers of rules — given that in popular P2P systems, such as Kazaa and Gnutella, it sometimes takes minutes for a search request to complete. As with the random topology, increasing the data replication, p_s, increases the average update response time. However, in the case of HyperCup, the system still remains stable and the update response time within acceptable boundaries even for high values of p_s. For example Figure 4 illustrates the case of $p_s = 90\%$.

Figure 5 shows how the update response time varies with the HyperCup topology as the number of rules per rule base, n_{rules} increases, with the data replication p_s being set to 10% and the number of peergroups set to 10, 50 and 100. We see that the system again shows good scalability, with the update response time increasing linearly with n_{rules}.

5.2 Simulation Results

As well as developing the performance model discussed above, we have also developed a simulator of the RDFTL system in order to increase the accuracy of the performance study and validate the predictions of the analytical model. This simulator was built using the Java implementation of the SSF (Scalable Simulation Framework) API [31], called Raceway SSF [27]. Raceway SSF provides a unified interface for discrete-event simulation, consisting of a set of classes for modeling the system entities, simulation events, communication channels and processes according to the discrete-event paradigm. The peers, superpeers, rule bases and rules are the main entities of our simulator. Peers and superpeers exchange messages representing events that can potentially affect the system state. For example, a message sent from a peer to a superpeer notifying the superpeer of the occurrence of an update at the peer may affect the state of the rule base, causing rule triggering that may result in more updates being executed, and so forth.

The same set of experiments performed on the analytical model were performed with the simulator. In these experiments, the same values for bps, k, p_{reduct} and $p_{mt}(0)$ were used as for the analytical model experiments, as shown in Table 1. The remaining system parameters

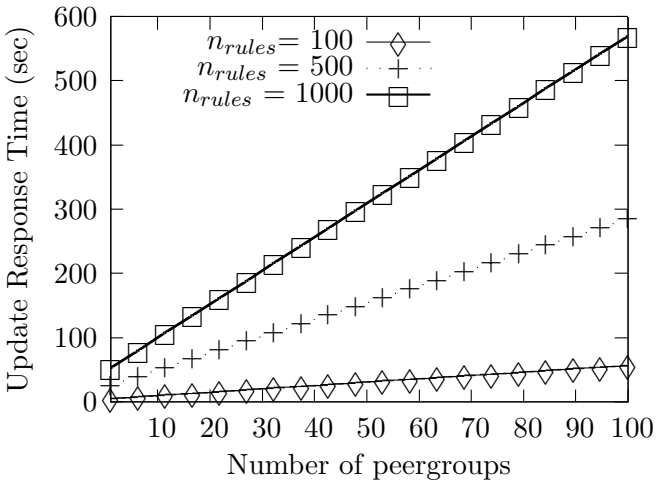

Figure 4: Model, Replication 90%, HyperCup

of Table 1 were replaced by stochastic values in order to provide a more realistic description of the system. In particular, the number of peers per peergroup, m, is randomly distributed with a mean value of 20; the probabilities p_{allow}, p_t and p_f are normally distributed, with mean values as in Table 1; and the remaining parameters are exponentially distributed, with mean values as in Table 1. Similarly, in the experiments, the number of rules per rule base, n_{rules}, and the degree of replication, p_s are not fixed but are normally distributed about a chosen mean.

Each of the data points in the graphs below was obtaining by running the simulator for 2000 top-level updates and taking the average update response time. The experiments were again conducted with the random and with the HyperCup network topologies.

Figure 6 shows how the update response time varies with the random topology as the number of peergroups, n, increases, with the data replication p_s distributed about a mean value of 10% and the number of rules per rule base, n_{rules}, distributed about mean values of 100, 500 and 1000. We see that the trend of these graphs is similar to Figure 2 from the analytical study, and it indicates that the system does not scale well with increasing n. Similar sets of experiments conducted for higher average values of p_s (up to 90%) result in graphs with similar upwards trends, except that the absolute values of the update response time are now larger and the system becomes unstable at lower values of n.

Results from the same set of experiments using the simulator and assuming a HyperCup topology are shown in Figures 7 and 8. We see that the system now shows good scalability, with the update response time increasing approximately linearly with n. With data replication at 10%, the update response time is within reasonable boundaries even for relatively large networks and numbers of rules. The system appears to scale well even for higher levels of metadata replication as illustrated in Figure 8 for data replication at 90%.

Finally, Figure 9 shows how the update response time varies with the HyperCup topology as the average number of rules per rule base, n_{rules} increases, with the data replication p_s being set to an average of 10% and the number of peergroups set to 10, 50 and 100. We see that the system again shows good scalability, with the update response time increasing linearly with n_{rules}.

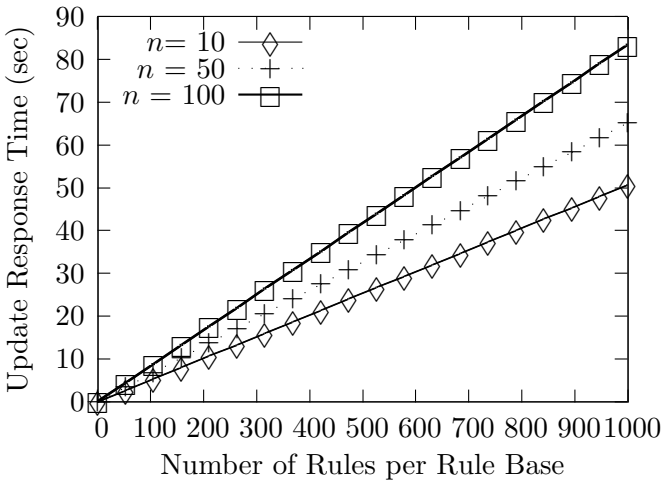

Figure 5: Varying Rules Model, Replication 10% , HyperCup

6 Conclusions and Future Work

This paper has studied the performance and scalability aspects of processing ECA rules on RDF metadata in P2P environments. We have developed and described in detail an analytical performance model for an implemented rule processing system for ECA rules on RDF metadata in P2P environments. We have used as the main performance criterion of the system the time required to complete all rule processing resulting from a top-level update submitted to one of the peers in the network. We have examined how this time varies with the network topology, number of peers, number of rules, and degree of metadata replication between peers. We have also described a simulation of the system, and have conducted similar performance and scalability experiments with the simulator. The two sets of experimental results show good agreement, which is an indication of the validity of the analytical performance model.

Both sets of experiments show that the system performance is significantly reliant on the network topology between superpeers. This is because P2P ECA rule processing is significantly affected by the number of messages exchanged between superpeers. Exploiting the HyperCup topology to reduce the number of messages improves the system's overall performance and scalability. With a HyperCup topology being used for interconnecting the superpeers, ECA rule processing on RDF metadata in P2P environments shows good scalability, pointing to its practical usefulness as a technology for real applications.

Fully replicating all superpeers' indexes at every superpeer would further reduce the number of messages exchanged during P2P ECA rule execution, by enabling the routing of messages only to the necessary superpeers. However, these indexes would need to be maintained as schema and metadata changes occur within peergroups. Studying the relative trade-offs involved is an area of future work.

To our knowledge, this is the first time that a P2P ECA rule processing system has been studied from a performance perspective, employing both analytical and simulation methods. Although conducted in the context of rules operating on RDF, we expect that similar behaviour would occur for P2P ECA rules operating on other types of data e.g. XML, relational.

For the future we would like to conduct large-scale experiments with the actual system

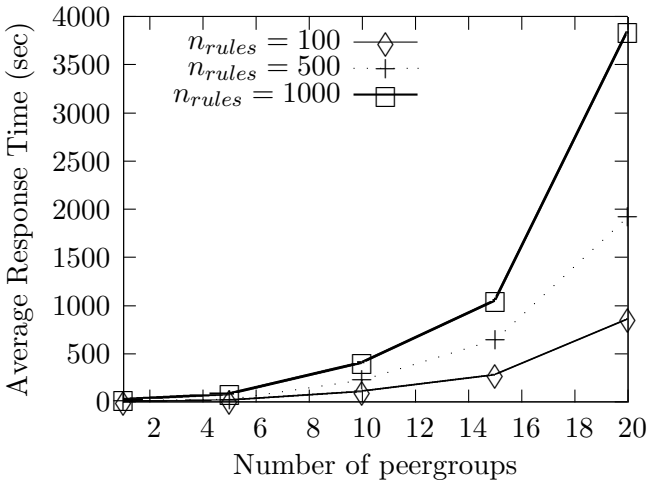

Figure 6: Simulation, Replication 10%, Full Net

itself, possibly using the PlanetLab [26] infrastructure. As well as giving insight into the actual system behaviour in a real P2P environment, this will allow measurements on actual system workloads and rule sets, which can then be fed into the analytical performance model and the simulator to allow more accurate predictions from these.

References

[1] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: peer-to-peer data and web services integration. In *Proc. VLDB'2002*, pages 1087–1090, 2002.

[2] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *Proc. SIGMOD 2003*, pages 527–538, 2003.

[3] J. Bailey, A. Poulovassilis, and P. T. Wood. Analysis and optimisation for event-condition-action rules on XML. *Computer Networks*, 39:239–259, 2002.

[4] J. Bailey, A. Poulovassilis, and P. T. Wood. An event-condition-action language for XML. In *Proc. WWW'2002*, pages 486–495, 2002.

[5] E. Baralis and A. Bianco. Performance evaluation of rule semantics in active databases. In *ICDE*, pages 365–374, 1997.

[6] A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active XQuery. In *Proc. ICDE'2002*, pages 403–418, 2002.

[7] A. Bonifati, S. Ceri, and S. Paraboschi. Active rules for XML: A new paradigm for e-services. *VLDB Journal*, 10(1):39–47, 2001.

[8] A. Bonifati, S. Ceri, and S. Paraboschi. Pushing reactive services to XML repositories using active rules. In *Proc. WWW'2001*, pages 633–641, 2001.

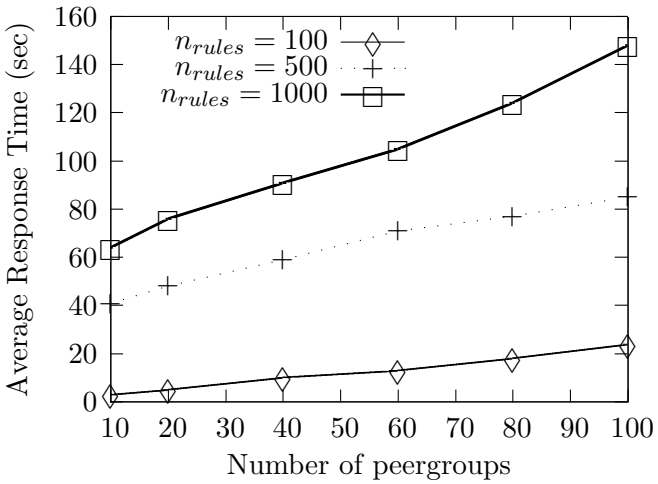

Figure 7: Simulation, Replication 10%, HyperCup

[9] A. Bonifati, U. Matrangolo, A. Cuzzocrea, and M. Jain. XPath lookup queries in P2P networks. In *Proc. 6th ACM Int. Workshop on Web Information and Data Management*, pages 48–55, 2004.

[10] S. Ceri, F. Daniel, V. Demaldé, and F. M. Facca. An approach to user-behavior-aware web applications. In *Proc. 5th Int. Conference on Web Engineering*, pages 417–428, 2005.

[11] P.-A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Publish/subscribe for RDF-based P2P networks. In *Proc. First European Semantic Web Symposium*, pages 182–197, 2004.

[12] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. *SIGCOMM Comput. Commun. Rev.*, 32(4):177–190, 2002.

[13] S. Gatziu, A. Geppert, and K. R. Dittrich. The SAMOS active DBMS prototype. In *Proc. SIGMOD'1995*, page 480.

[14] George Papamarkos and Alex Poulovassilis and Peter T. Wood. Event-Condition-Action Rules on RDF Metadata in P2P Environments. *Computer Networks*, October 2006.

[15] A. Geppert, S. Gatziu, and K. R. Dittrich. A designer's benchmark for active database management systems: 007 meets the BEAST. In *Proc. 2nd Int. Workshop on Rules in Database Systems*, volume 985, pages 309–326. Springer, 1995.

[16] ICS-FORTH. RDFSuite. `http://139.91.183.30:9090/RDF/`.

[17] R. Jain. *The Art of Computer Systems Performance Analysis.* Wiley, 1991.

[18] V. Kantere, I. Kiringa, J. Mylopoulos, A. Kemenstiestides, and M. Arenas. Coordinating peer databases using ECA rules. In *Proc. DBISP2P*, pages 108–133, 2003.

[19] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: a declarative query language for RDF. In *WWW'2002*, pages 592–603, 2002.

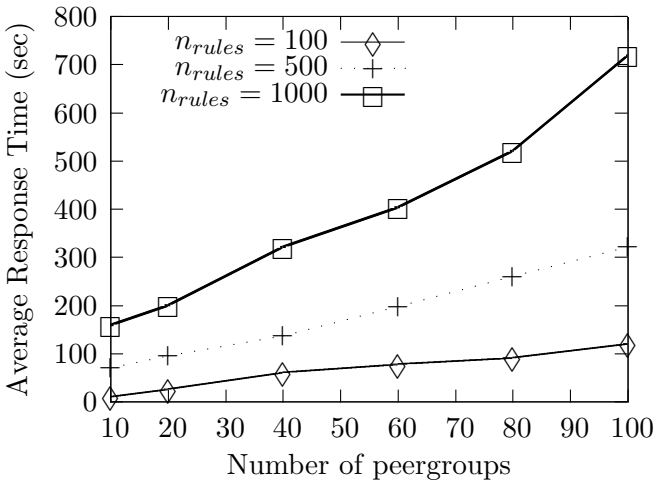

Figure 8: Simulation, Replication 90%, HyperCup

[20] G. Kokkinidis and V. Christophides. Semantic query routing and processing in P2P database systems: The ICS-FORTH SQPeer middleware. In *Proc. EDBT Workshops*, pages 486–495, 2004.

[21] M. Magiridou, S. Sahtouris, V. Christophides, and M. Koubarakis. RUL: A declarative update language for RDF. In *Proc. 4th Int. Semantic Web Conference*, pages 506–521, 2005.

[22] W. Nejdl, W. Siberski, B. Simon, and J. Tane. Towards a modification exchange language for distributed RDF repositories. In *Proc. 1st Int. Semantic Web Conference*, pages 236–249, 2002.

[23] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *Proc. WWW'2002*, pages 604–615, 2002.

[24] M. Nicola and M. Jarke. Performance modeling of distributed and replicated databases. *IEEE Trans. on Knowledge and Data Engineering*, 12(4):645–672, 2000.

[25] N. Paton. *Active Rules in Database Systems*. Springer, 1999.

[26] PlanetLab. `http://www/.planet-lab.org`.

[27] Raceway SSF. `https://gradus.renesys.com/exe/Raceway`.

[28] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP – hypercubes, ontologies and efficient search on P2P networks. In *First Int. Workshop on Agents and P2P Computing*, volume 2530 of *LNCS*, pages 112–124. Springer, 2002.

[29] W. Siberski and U. Thaden. A simulation framework for schema-based query routing in P2P networks. In *Proc. Workshop on Peer-to-Peer Computing and Databases (in conjunction with EDBT)*, pages 436–445, 2004.

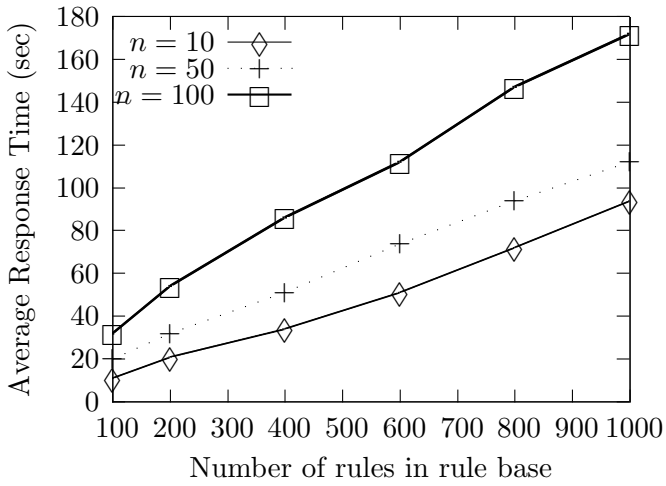

Figure 9: Varying Rules Simulation, Replication 10%, HyperCup

[30] Simple Sharing Extensions (SSE) for RSS and OPML. `http://msdn.microsoft.com/xml/rss/sse/`.

[31] SSF — Scalable Simulation Framework. http://www.ssfnet.org/homePage.html.

[32] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The Piazza Peer Data Management Project. *SIGMOD Record*, 32(3):47–52, 2003.

[33] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proc. of the 2nd Int. Workshop on Distributed Event-Based Systems*, pages 1–8, 2003.

[34] W3C. XML Path Language (XPath), 1999.

[35] W3C. RDF Semantics, W3C Recommendation 10 February 2004, 2004.

[36] W3C. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004, 2004.

[37] W3C. RDF/XML Syntax Specification, W3C Recommendation 10 February 2004, 2004.

[38] J. Widom and S. Ceri. *Active Database Systems*. Morgan-Kaufmann, San Mateo, California, 1995.