Engineering Applications of Artificial Intelligence 24 (2011) 1487–1498

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

JTangCSPS: A composite and semantic publish/subscribe system over structured P2P networks

Jianfeng Qian, Jianwei Yin*, Jinxiang Dong, Dongcai Shi

School of Computer Science and Technology, Zhejiang University, 310027 Hangzhou, China

ARTICLE INFO

Available online 4 May 2011

Keywords:
Semantic technology
Composite event
Semantic event
Publish/subscribe
P2P network

ABSTRACT

Publish/subscribe systems offer a loosely coupled communication paradigm in distributed information systems. However, supporting expression of semantic events, expression of logical and temporal patterns of composite events, and how to manage and route composite events and subscriptions still need further research. In this paper, we present the design and implementation of JTang composite and semantic publish/subscribe system over structured P2P networks, and highlight its novel features, including semantic broker network, composite event and subscription language and distributed composite subscription management. The experiments based on the Peersim simulator over the Pastry overlays show that the ontology routing table helps decrease the average number of hops and the use of composite subscriptions significantly reduces the load on the network.

1. Introduction

Publish/subscribe systems (Eugster et al., 2003) offer a loosely coupled communication paradigm in distributed information systems. A traditional publish/subscribe system is composed of publishers, subscribers and brokers. Publishers generate information in the form of event and sent it to a network of connected brokers instead of subscribers directly. In order to receive events, subscribers need to register their interests with brokers in the form of subscription. Brokers are responsible for managing subscriptions and delivering the events to the interested subscribers through the network.

Content-based publish/subscribe systems, which allow subscribers to specify the kind of message content they want to receive, have gained much attention for their powerful expressiveness in the distributed computing environment. A large variety of emerging applications including RSS feed filtering, stock-market monitoring engines, system and network management and monitoring benefit from content-based publish/subscribe systems.

However, content-based publish/subscribe systems are not aware of the semantics of information. For example, suppose a subscriber is interested in information about "computer", usually will not receive an event about "notebook". Ontology languages such as OWL or RDFS enable building correlations based on inherited meanings of orthogonal concepts as well as on relationships between them. Using ontological concepts and relations to express events and subscriptions can improve the expression of publish/subscribe system, and increase selectivity and finer grained control for the event matching process. Moreover, semantic reasoning can discover additional events that cannot be found by traditional matching algorithms.

On the other side, more sophisticated publish/subscribe applications such as business process execution, business activity monitoring and workflow management need composite subscriptions to combine events from distributed sources. Composite events can be composed or derived from other events called its members. Composite subscriptions allow clients to subscribe patterns of events (composite events). It gives additional dimension of data management, and improves scalability and performance of distributed systems.

However, the issues of how to support semantic events and logical and temporal correlations of composite events both and how to route and manage composite events and subscriptions are still largely unexplored.

To address the above issues, we present the design and the implementation of JTang Composite and Semantic Publish/subscribe system (JTangCSPS) over structured P2P networks. Compared with the existing work, the main contributions of our work are as follows:

1. We propose a Semantic broker network architecture, which is built by mapping ontology class weighted tree to weighted broker network. The ontology routing table maintains OCWT and helps deduce the hops, and the backup strategy improves the reliability of system.

* Corresponding author.
E-mail addresses: jfqian@163.com (J. Qian), zjuyjw@cs.zju.edu.cn (J. Yin), djx@cs.zju.edu.cn (J. Dong), shidcai@163.com (D. Shi).

0952-1976/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2011.03.006

2. We present a composite event and subscription language to support logical and temporal correlations among distributed events and subscriptions. An additional ontology is created to store composite event and subscription in OWL language.
3. A distributed composite subscription management is used to decompose composite subscriptions and collect and aggregate primitive events from the different brokers to satisfy the subscriptions.
4. A semantic matching algorithm and a semantic routing algorithm are presented to support the semantic matching and routing.

2. Related work

Recently, some publish/subscribe systems, such as DREAM (Buchmann et al., 2004), OPS (Wang, 2005), S-TOPSS (Petrovic et al., 2003), G-ToPSS (Petrovic et al., 2005), allow events to be matched to subscriptions based not only on their contents, but also on their semantics. Events and subscriptions express the semantics of their information based on an ontological representation of that information. DREAM (Buchmann et al., 2004) is a reactive event-based middleware platform, which develops concept-based notification to extend content-based filtering to heterogeneous environments. OPS (Wang, 2005) is an ontology-based publish/subscribe system and its event model and subscription model are based on RDF. It offers a subscription language, which is a subset of the DAML language (McGuinness et al., 2002) and supports hierarchical and equivalent class as well as hierarchical property relations. S-ToPSS (Petrovic et al., 2003) uses an ontology that includes synonyms, taxonomy and transformation functions to deal with syntactically disparate events and subscriptions. G-ToPSS (Petrovic et al., 2005) is an RDF-based publish/subscribe system developed for selective information dissemination. It utilizes the ontology to interpret semantic information about the data and a fast graph-based matching algorithm to filter semantic events. However, these publish/subscribe systems restrict subscriptions to primitive events. To the best of our knowledge, JTangCSPS is the first system, which provides a composite event and subscription language for broadly supporting the semantic events and subscriptions and logical and temporal composite operators.

A promising solution for the design and deployment of large-scale publish/subscribe system is the exploitation of the peer-to-peer (P2P) paradigm. P2P networks can be categorized into two main types: unstructured and structured. Structured P2P networks offer characteristics such as efficient routing, key-search, self-organization, fault tolerance and good load balancing by a Distributed Hash Table (DHT). Pietzuch and Bacon (2002), Baldoni et al. (2005), Ahull et al. (2008) and Pujol-Ahullo et al. (2009) are content-based publish/subscribe over the DHT overlay and employ a rendezvous model to facilitate meetings between events and subscriptions. Rendezvous brokers are responsible for storing subscriptions, matching events and notification process. However, the “hash-like” interaction is not sufficient for a content-based publish/subscribe system that requires support for more complex and expressive queries. Besides, Chirita et al. (2004) built a publish/subscribe system on the Edutella (Nejdl et al., 2003) P2P infrastructure, which use super-peer based topologies and organized peers in hyper-cubes. However, queries have to be flooded to every broker node in the network, making the system difficult to scale. Crespo and Garcia-Molina (2003) proposed the concept of Semantic Overlay Networks (SONs). SONs (Crespo and Garcia-Molina, 2003; Löser et al., 2003; Schmitz, 2004; Raftopoulou and Petrakis, 2008) are an instance of unstructured networks in which peers are grouped by thematical, semantic or social relationships of documents they store. Each peer stores additional information about content classification and route queries to the appropriate SONs, increasing the chances that matching objects will be found quickly and reducing the search load. However, the maintenance cost in SONs becomes more expensive when the number of SONs increases. Different from these systems, JTangCSPS proposes a semantic broker network over structured P2P network to achieve large-scale computing and load balance.

Composite events have been an important issue not only in traditional networks such as system monitoring but also in distributed event systems. Some publish/subscribe systems support composite events and subscriptions. Courtenage (2002) describes a new declarative language for specifying composite events based on the typed λ-calculus. Composite events are represented in this language by curried functional expressions. Following research (Courtenage and Williams, 2006) implements a composite event notification system over a chord-based peer-to-peer network using JXTA to handle the network management and network routing. CEA (Pietzuch et al., 2003; Pietzuch et al., 2004) proposes a Core Composite Event Language to express concurrent event patterns. The CEA language is compiled into automata for distributed event detection supporting regular expression-type patterns. CEA employs polices to ensure that mobile event detectors are located at favorable locations, such as close to event sources. REBECA (Ulbrich et al., 2004) describes composite events using composite event filter expressions, which can be mapped to expressions of the Core Composite Event Language (Pietzuch et al., 2003). ECCO (Yoneki and Bacon, 2005) created event broker architecture for a distributed adaptive mobile environment. Event correlation is part of event brokers, and grids of brokers are deployed over mixed network environments. The ECCO prototype is implemented with a MANET protocol as a content-based publish/subscribe system. Cayuga (Demers et al., 2006; Demers et al., 2007) is a stateful publish/subscribe system based on nondeterministic finite state automata (NFA), and supports features such as parameterization and aggregation. Gang et al. (2005) proposes the composite matching algorithm based on hierarchy colored Order Binary Decision Diagram (OBDD) graphs to handle temporal constraint variable of the composite subscription. PADRES (Li and Jacobsen, 2005) supports parallelization, alternation, sequence and repetition compositions, and is based on a rule-based broker that implements composite event detection and proposes a distributed algorithm for composite subscription routing. However, REBECA (Ulbrich et al., 2004) allows subscribing temporal events only and (Courtenage, 2002) lacks support for temporal correlation. Some other systems gave no distributed solutions (Demers et al., 2006; Demers et al., 2007; Gang et al., 2005), and Yoneki and Bacon (2005) did not consider efficiency. In this paper, we present a composite event and subscription language and distributed composite subscription management over semantic broker network to achieve large-scale computing and efficient routing.

3. Semantic broker network architecture

Distributed publish/subscribe systems strive to achieve scalability and avoid a single point of failure using an overlay network of brokers. However, the hash function of structured P2P network is not sufficient for a content-based publish/subscribe system. In this section, we present the detail of the semantic broker network architecture. JTangCSPS is an extension of our early work (Qian et al., 2011). Web Ontology Language (OWL) is used to build semantic broker network. We define weights of ontology classes to estimate the information of events and subscriptions, and create an ontology class weighted tree (OCWT) based on the inheritance and similarity of ontology classes. Next, OCWT is mapped to a one-dimensional space and each ontology class of OCWT is assigned a unique numeric ontology class identifier (OCId). We also define the weights of brokers to present the capability of the brokers. Then, the OCWT is mapped to the weighted broker network. Brokers with higher weights are mapped with ontology classes with higher weights, and one or more ontology classes that are connected in OCWT might be mapped to the same broker. Subscriptions related to the same ontology class are defined as a virtual subscription, events and subscriptions are routed to different brokers, who manage the responding virtual subscriptions. By this approach, system achieved semantic-based division of the structured P2P network, large-scale distributed computing capabilities and load balance.

3.1. Ontology class weighted tree

To quantitatively estimate the information of events and subscriptions, we define the weight of OWL ontology class C_x by

$$W_{Cx} = \sum_{k=1}^{K} W_{Cy_k} + \sum_{l=1}^{L} W_{DP_l} + \sum_{m=1}^{M} W_{OP_m} W_{C_m} \tag{1}$$

W_{DP} and W_{OP} are the weights of datatype property and object property of class C_x, W_{Cm} is the weight of the class of the range of the W_{OP}, L and M are the numbers of W_{DP} and W_{OP}, K is the number of the parents of class C_x. To simplify, the weight of property is set to 1. In the case of single inheritance, the formula becomes

$$W_{Cx} = W_{Cy} + \sum_{l=1}^{L} W_{DP_l} + \sum_{m=1}^{M} W_{OP_m} W_{C_m} \tag{2}$$

We create an ontology class weighted tree (OCWT) based on the inheritance and similarity of ontology classes. The OCWT is mapped to a one-dimensional space and each ontology class of the OCWT has a unique numeric ontology class identifier (OCId) by the OCId algorithm. When OCWT is mapped to the 2^{bl} one-dimensional space, we assume the inequalities ($2^b - 1 >= n$ and $bh < l - 1$) are satisfied where h is the height of the OCWT and n is the max number of children of the OCWT.

```
OCId Algorithm
Input: OCWT
Output: COCWT
root.OCId = $10\ldots0_{l-2}$
for $cn_i \in$ OCWT
  if $cn_i$.CS.size $> 0$
    $ws = \sum_{j=1}^{cn_i.CS.size} cn_i.CS.get(j).weight$
    $cn_i$.CS.sort ()
    for($j = 1 \ldots cn_i$.CS.size)
      $cn_i$.CS.get($j$).OCId = $cn_i$.OCId
 $cn_i$.CS.get($j$).OCId.$x$ = $\begin{cases} 1 & j = 1 \\ x_f + \min([(2^b - 1) * w_f / ws], 2^b - (cn_i.CS.size - i)) & j > 1 \end{cases}$
```

The OCId algorithm starts from the root of the OCWT, the OCId of root is a numeric value represented as a single '1' followed by l–2 '0's. For each class node cn_i of OCWT, if the size of the children set CS of cn_i is positive, the sum of the weights of all children is assigned to ws. Children nodes are sorted by similarity $S_{Cx,Cy}$ defined by formula 3, where the numerator is the weight of properties that both classes have and the denominator is the average weight of Cx and Cy. The child that is most similar to the parent is put in the first place of sorted children set CS, and the sibling that is most similar to the first child is put in the next place and so on. The OCId of the child node is assigned with that of parent first, then set the first 0 (x bit) to a value between 1 and $2^b - 1$ according to its weight. The x bit of the first child is set to 1, and the x bits of the other children are set by the equation in OCId algorithm where x_f is the x bit value of the former child and w_f is the weight of the former child. The equation guarantees that every ontology class has a unique OCId.

$$S_{Cx,Cy} = \frac{\sum_{l=1}^{L'} W_{DP_l} + \sum_{m=1}^{M'} W_{OP_m} W_{C_m}}{(W_{Cx} + W_{Cy})/2} \tag{3}$$

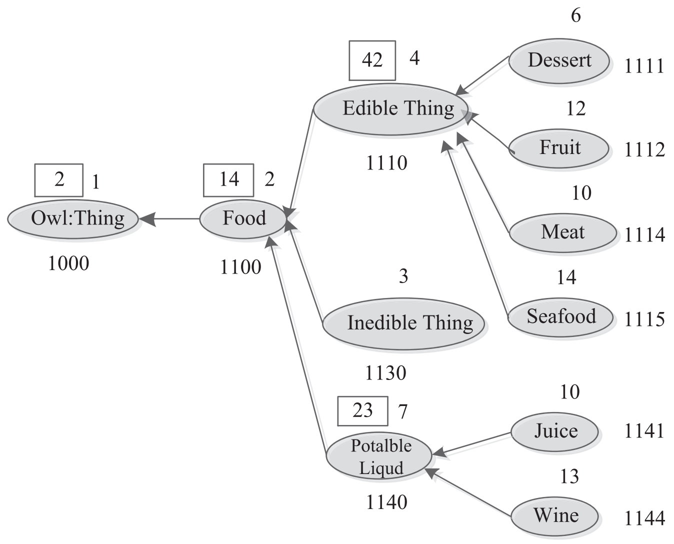

Fig. 1. OCWT and OCId.

Fig. 1 is an example of the ontology of food. The number above the node is the weight of ontology class, the number in the rectangle is the sum of the children's weights, and the number behind the node is the value of OCId in the case of $b=3$, $l=4$.

3.2. Weighted broker network

For distributed publish/subscribe systems, the most important resources are the bandwidth of the network and computing ability of the brokers. Besides, brokers usually serve continuously. We define the weight of the brokers by

$$W_N^t = a_1 B + a_2 C + a_3 t_{online} \tag{4}$$

W_N^t is the weight of the broker at t time, B is the bandwidth of the broker, C is the computing ability, t_{online} is the online time and a_1, a_2 and a_3 are constants.

After defining the weight of the broker, we map OCWT to the weighted broker network. To achieve load balance, we let the bigger weight of broker responsible for the bigger weight of ontology classes. Subscriptions related to the same ontology class are defined as a virtual subscription, and each broker manages one or more virtual subscriptions.

When the number of nodes of OCWT is larger than the number of nodes of the broker network, some brokers will be responsible for one or more ontology classes. We combine the class nodes of OCWT to create a new tree named composite ontology class weighted tree (COCWT). Correspondingly, a composite ontology class contains one or more ontology classes that are connected (parent, left and right sibling and children).

```
OCWT Combination Algorithm
Input: OCWT
Output: COCWT
combine(OCWT)
if $n < m$
  for $cn_i \in$OCWT
    if all children of $cn_i$ are leaves
      LBS.add($cn_i$)
  for $lb_i \in$ LBS
    pair=createPair($lb_i$)
    PS.add(pair)
  for ($i$=1…$m$-$n$)
    pair=PS.get(0)
    ccn.weight=pair.weight
    ccn.NS.add(pair.a)
    ccn.NS.add(pair.b)
    pair.a.parent.CS.remove(pair.a)
    if pair.b.parent==pair.a
      pair.a.CS.remove(pair.b)
    else
      pair.a.parent.CS.remove(pair.b)
    pair.a.parent.CS.add(ccn)
    if pair.b.parent==pair.a
      LBS.add(pair.a.parent)
      PS.remove(pair.a)
      PS.add(createPair(pair.a.parent))
  createPair(lb)
  if lb.CS.size==1
    pair.a=lb
    pair.b=lb.CS.get(0)
    pair.weight=pair.a.weight+pair.b.weight
  else
    pair.weight=MAX_INTEGER
    for($i$=1…lb.CS.size-1)
      sum= lb.CS.get($i$-1).weight+ lb.CS.get($i$).weight
      if pair.weight > sum
        pair.weight=sum
        pair.a=lb.CS.get($i$-1)
        pair.b=lb.CS.get($i$)
```

n is the number of broker nodes and m is the number of ontology class tree nodes. If m is larger than n, those nodes whose children leave are put into set LBS. Then, for each node of LBS, a pair is created and put into pair set PS sorted by the weight of pair. Each pair has two nodes, which are parent node and the only child, or two connected children nodes with the smallest sum of weights. Each time, the broker gets the first pair from PS to create a composite class node ccn, and pair.a and pair.b are put into nodes set NS of ccn, then ccn is put into the children set CS of parent of pair.a, and pair.a and pair.b are removed from the OCWT. If pair.a is the parent of pair.b, parent of pair.a is put into set LBS and a new pair is created and put into PS while the old pair is removed from PS. The loop will continue until the number of nodes of the COCWT is equal to n.

Due to lack of the complete information of the broker network at the beginning, the OCWT cannot be mapped to the broker network directly. First, a broker node is selected as root node randomly, then root node sends request messages to the nodes in the routing table to collect the weights of the broker nodes, and the OCWT is combined into the COCWT until the number of composite ontology class is equal to the number of the nodes in the routing table. Finally root node arranges the weights of the composite ontology class and broker nodes and assigns one composite ontology class to one node in order. If a broker node is mapped with a composite ontology class, it gets the responding subtree from the OCWT and tries to split it and maps them to nodes in its routing table.

```
Mapping Algorithm
Input: OCWT
Output:
map(OCWT)
for $node_i \in$routing table
  WS.add( request($node_i$))
WS.sort
combine(OCWT)
for $node_i \in$ WS
  $node_i$.map(OCWT.$ccn_i$)
```

3.3. Ontology routing table and backup strategy

The semantic broker network has three types of brokers: main node, backup node and routing node. The main node manages one or more virtual subscriptions, and the backup node backs up the virtual subscriptions of one main node, while a routing node only helps to route messages. Each broker has a node identifier (NId) by combining the OCId and the broker identifier (BId). The NId is used to indicate a broker's position in a circular NId space, and BId is generated by hashing its IP address to k-bit space; so the length of NId is $bl+k$. For a main node, OCId is the smallest one of the virtual subscriptions. For a backup node, OCId is same as the main node it backs up. For a routing node, OCId is same as the node that the broker joins the network and first connects to.

Besides the routing table of the understructure P2P network, each main node has an ontology routing table (ORT), including NodeSet, ParentSet, ChildrenSet, GrandChildrenSet, LeftNeighbor, RightNeighbor, BackupNode and BebackupNode. NodeSet contains the OCIds of the ontology classes, which the broker maintains; ParentSet contains the OCIds of parents and grandparents of those in the NodeSet; ChildrenSet contains the OCIds of all children of those in the NodeSet; GrandChildrenSet contains the OCIds of first child of those in the ChildrenSet; LeftNeighbor and RightNeighbor are the OCIds of the left and right siblings; BackupNode is the OCId of the node, which serves as a backup node of this node and BebackupNode is the OCId of the node, which is backed up by this node. ORT helps route messages and reconstruct the COCWT.

To improve the reliability of system, a backup strategy is used. During selection of the backup node, we consider two situations: (1) if $n \le m$, then each broker is a main node. How to select the backup node is decided by the following conditions: (a) parent node is backed up by the first child; (b) leaf node is backed up by the left or right neighbor node or the parent node. It ensures that the node and backup node are always adjacent. (2) If $n > m$, a main node try to find a not main node from the routing table as a backup node. If it cannot find one, it will follow the method above.

Fig. 2 is an example of mapping the COCWT of Fig. 1 to Pastry (Rowstron and Druschel, 2001) network, which has eight nodes. N_{1000} is the root of the COCWT, and N_{1111}, N_{1114} and N_{1141} are the composite nodes. Arrows with dotted line connect parents to their children.

```
Routing algorithm
Input: Message $M$
Output: next hop
getNextHop($M$)
if ($L_{-l/2} \le M.dest \le L_{l/2}$)
  return min($|M.dest - L_i|$)
```

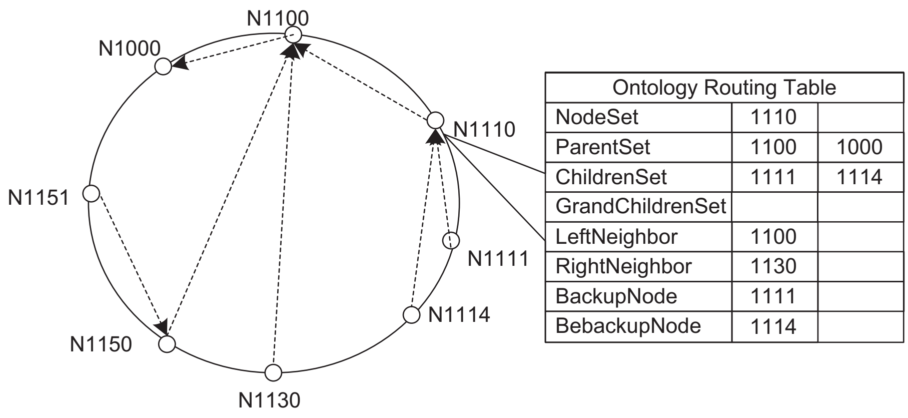

Fig. 2. Semantic Pastry broker network.

```
else
  $l = \mathrm{shl}(M.dest, \mathrm{OCId})$
  **if** $M.dest < \mathrm{OCId}$
    $k = \max(\mathrm{shl}(M.dest, \mathrm{ORT.ParentSet}_i))$
    **if**$(k > l)$
      **return** $\mathrm{ORT.ParentSet}_i$
  else
    $\mathrm{Set} = \mathrm{ORT.ChildrenSet} \cup \mathrm{ORT.GrandchildrenSet}$
    $k = \max(\mathrm{shl}(M.dest, \mathrm{Set}_i))$
    **if**$(k > l)$
      **return** $\mathrm{Set}_i$
  **else if**($R_l^{DI}$ is not null)
    **return** $R_l^{DI}$
  else
    $NH = L \cup R \cup M \cup \mathrm{ORT}$
    **if** $\mathrm{shl}(M.dest, NH_i) \geq 1$ and $(|M.dest, NH_i| < (M.dest, \mathrm{OCId}))$
      **return** $NH_i$
```

In the semantic Pastry broker network, each main node maintains a neighborhood set, a leaf set, a routing table and ontology routing table. The routing process is executed whenever a message (event or subscription or notification) arrives at a broker. Given a message, the destination is determined by the ontology class it belongs to, the broker first checks to see if the destination is in the NodeSet of ontology routing table. If so, the broker handles the message correspondingly. Then, the broker checks if the destination falls within the range of OCIds covered by its leaf set. If so, the message is forwarded directly to the destination broker, namely the broker in the leaf set whose OCId is closest to the destination. If the destination is not covered by the leaf set, different form Pastry, the broker checks ontology routing table. The destination of message is compared with the OCId of the broker. If the destination is smaller than the OCId, a candidate is picked from the ParentSet of ORT who shares the maximum common prefix with the destination of message, else a candidate is picked from the ChildrenSet or GrandChildrenSet of ORT in the same way. If the next hop is still not found, broker picks a candidate broker from the routing table by the default way, and the message is forwarded to the broker that shares a longer common prefix with the destination. The function shl(A,B) returns the length of the prefix shared among A and B in digits, which is mentioned in Rowstron and Druschel (2001). ParentSet, ChildrenSet and GrandChildrenSet of ORT serve as shortcuts to reach the destination; so hops of semantic Pastry will be less than or at least equal to the hops the Pastry ($\log_2 {}^bN$).

4. JTangCSPS composite event and subscription language

In a distributed system, each event has a timestamp associated with the occurrence time. There are two types of expressions of timestamps: point-based timestamps and interval-based timestamps. Point-based timestamps usually consist of a single value, indicating the occurrence time. However, in a distributed system, node clocks may have unknown jitter within a known synchronization distance. As a result, if two nodes detect events E_1 and E_2, it may be impossible to decide which occurred first. On the other side, interval-based timestamps consist of a start time and an end time, which can make this ambiguity explicit and remain consistent with the physical time order of events. Interval-based timestamps are used in this paper.

In this section, we present a composite event and subscription language, and show the presentation of composite event and subscription model. Then we illuminate the management of distributed composite subscriptions including subscribing process, unsubscribing process and matching and notifying process.

4.1. Composite event and subscription language

To describe formally primitive and composite events (subscriptions), we adopt a symbolic representation that resembles and extends the CEA language (Pietzuch et al., 2003) and is conformant with the logical and temporal expression of Lamport (1978) and Allen and Ferguson (1994).

Events and subscriptions in JTangCSPS are related to domain knowledge and defined by OWL ontology language. A primitive event E_p is defined by the following formula:

$$E_p = (c, S_{tr}, t, t_l) \tag{5}$$

$$S_{tr} = \{(s_1, p_1, o_1), \ldots (s_n, p_n, o_n)\} \tag{6}$$

$$t = (t^s, t^e) \tag{7}$$

E_p is an individual of a class $c \in C$ (the classes of ontology) and is associated with a set of triples S_{tr} and an interval timestamp $t \in T$ and a lifetime t_l. S_{tr} is composed with triples, which have a subject s, a property (predicate) p and an object o. t^s is the interval start time, and t^e is the interval end time. t_l is the lifetime of the event and has a finite value.

Composite events are defined by composing primitive events or composite events with logical and temporal operators. A composite event can be created by publishers or combined by brokers to satisfy a composite subscription. A composite event E_c

is defined by the following formula:

$$E_c = (op(E_1,\ldots,E_n),t) \tag{8}$$

$$t^s = \min(t_1^s,\ldots,t_n^s) \tag{9}$$

$$t^e = \max(t_1^e,\ldots,t_n^e) \tag{10}$$

E_i can be a primitive event or a composite event. The interval start time t^s of E_c will be the earliest start time of the constituent events, and interval end time t^e of E_c will be the latest end time of the constituent events. The operators supported by JTangCSPS will be introduced later.

A primitive subscription S is defined by the following formula:

$$S = (c,S_{tr},F,t,t_l) \tag{11}$$

$$F = \{(s_1,p_1,mop_1,o_1),\ldots,(s_n,p_n,mop_n,o_n)\} \tag{12}$$

Different from the primitive event, primitive subscription S has an additional set of filters F. Filter is different from triple for it has a mathematical operator mop as $=$, $<$, $>$, $\le$ or $\ge$.

A composite subscription S is defined by the following formula:

$$S_c = (op(S_1,\ldots,S_n)) \tag{13}$$

A composite subscription S_c is similar to the composite event but without an interval timestamp.

Next, we will present operators supported by JTangCSPS in examples of composite subscriptions. For the sake of convenience, S is defined as a subscription, E as an event, N as a notification and E_i as the event matched with S_i. The operators will be present in a more easy understandable way rather than defined above:

- Temporal restriction S_T: Event E occurs within interval T.
- Conjunction. S_1 and S_2: Events E_1 and E_2 occur in any order and without time restriction. $(S_1 \text{ and } S_2)_T$: E_1 and E_2 occur in any order but one occurs within the T interval time of the other.
- Disjunction. $S_1|S_2$: Event E_1 or E_2 occur without time restriction. $(S_1|S_2)_T$: E_1 or E_2 occurs, if one occurs within the T interval time of the other, a composite event (E_1 and E_2) will be created.
- Concurrency. $S_1||S_2$: Events E_1 and E_2 occur in parallel.
- Negation $(-S_1)_T$: No event E_1 occurs within the T interval and a notification message will be created. $(S_1-S_2)_T$: E_1 occurs and no E_2 occurs within the T interval. The negation operator of subscription is always followed by a T restriction, otherwise there will be no notification.
- Iteration S_1^n: A number of n event E_1 occurrences $(S_1^n)_T$: A number of n event E_1 occurrences with each in the T interval of the former E_1. $(S_1^*)_T$: Any number of event E_1 occurrences with each in the T interval of the former E_1.
- Sequence. It includes 7 temporal restrictions: S_1 before S_2, S_1 meets S_2, S_1 overlaps S_2, S_1 finishes S_2, S_1 includes S_2, S_1 starts S_2, S_1 is equal to S_2. $S_1{=}S_2$ is same as $S_1||S_2$. The corresponding sequence relation of E_1 and E_2 (Nejdl et al., 2003) is shown in Fig. 3.

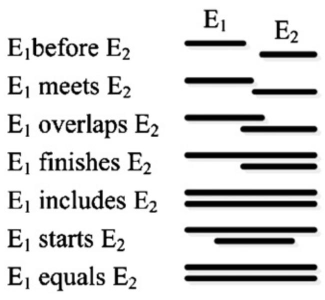

Fig. 3. Temporal sequence of events.

4.2. Presentation of composite event and subscription model

In JTangCSPS, events and subscriptions are represented as RDF graphs. Since OWL does not support logical and temporal operators and quantified variables natively, a special ontology is created to present the concepts and relations of event and subscription. Fig. 4 is a part of the ontology about subscriptions. Both event and subscription have an additional property to express the interval-based timestamp, and the other ontology classes about events are similar.

The structure of an RDF graph is a collection of triples, and each triple consists of a subject, a property and an object. Every triple is denoted as a node–arc–node link in the RDF graph. A node may be a URI reference, a literal or blank, while an arc is always a URI reference and points toward the object. Fig. 5 is an example of RDF primitive event graph. The value of *rdf:type* property means that it is a primitive event individual and it has a *hasTimestamp* property. The value of *hasInfo* property can be any individual and *rdf:type* property identifies the ontology class, which the event belongs to, and the other properties are presented as p_i; URI reference and text are presented as *URIref_i* and *Literal_i*($i \ge 1$).

Fig. 6 illustrates a primitive subscription S built on the event graph E. The literal of p_1 started with the operator " > " to support traditional mathematical operation. Fig. 7 illustrates that a composite subscription has two primitive subscriptions with a temporal operator "before".

4.3. Distributed composite subscriptions management

In a large-scale distributed publish/subscribe system, events and subscriptions are sent to different brokers. A composite event

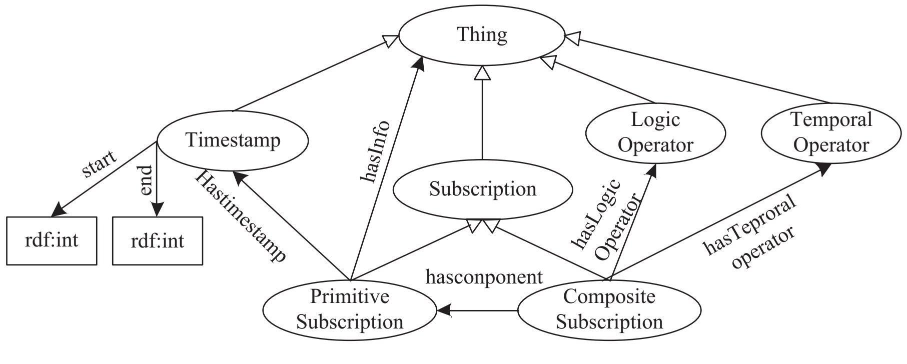

Fig. 4. Subscription ontology.

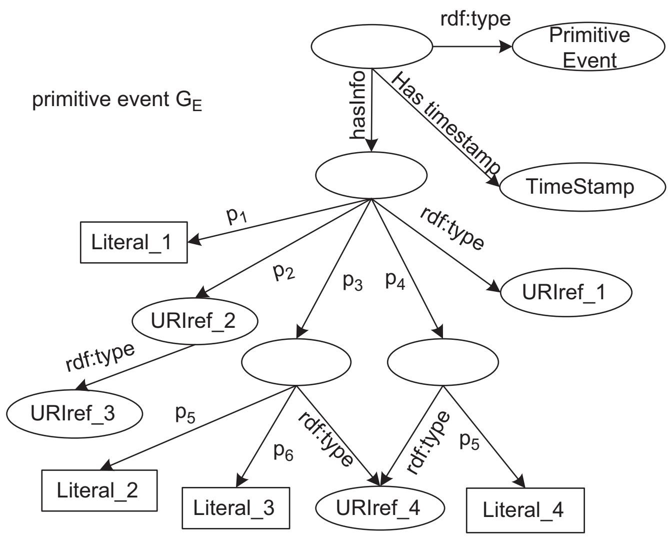

Fig. 5. Primitive event G_E.

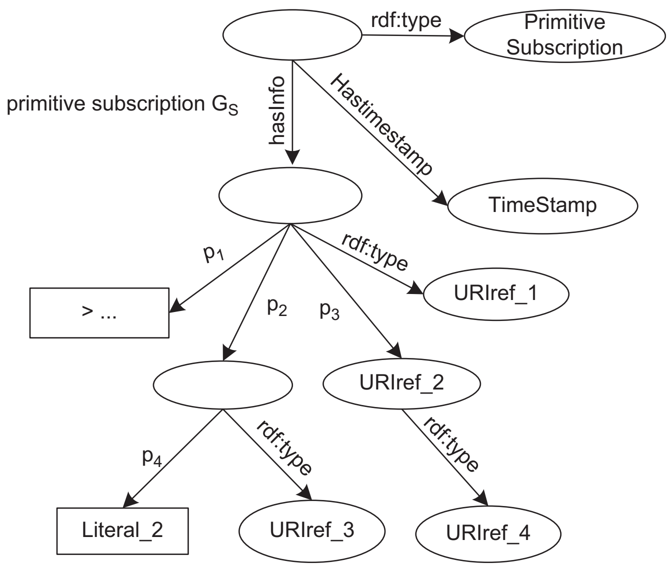

Fig. 6. Primitive subscription G_S.

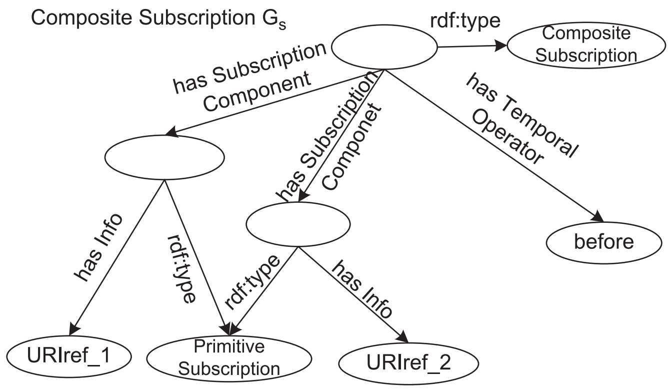

Fig. 7. Composite subscription G_S.

is created by broker when a sequence of events is matched with a composite subscription. The main difficulties of the routing algorithm are where and how to decompose a composite subscription and route the individual parts of the subscription. In this paper, we propose a distributed composite subscriptions management. The distributed solution consists in decomposing the composite subscription and registering the parts of the subscription to different brokers during the routing process, and collecting the primitive events from the different brokers and aggregating them through the reverse path of subscription routing. A notification message is sent to the subscriber only after all the parts are collected. The collections of events should be as close to the publishers as possible to ensure that the events are not unnecessarily disseminated throughout the broker network.

To store primitive and composite subscriptions, every broker has a primitive subscription map (PSM) and a composite subscription map (CSM). Each subscription has a unique subscription identifier (SId), which is used as the key of the two maps. During the subscribing process, when the broker receives a primitive subscription, it will be routed to the destination broker who manages the responding ontology class of the subscription, and then be put into the PSM. On the other hand, when a broker receives a composite subscription, a composite subscription tree (CST) is built by regular express. Leaf nodes of CST are primitive subscriptions and the internal nodes are operators. According to the OCId algorithm, each primitive subscription has its destination. If all primitive subscriptions have the same next hop, the broker forwards the composite subscription as a whole to the next hop; otherwise it is put into the CSM and the broker creates a new composite subscription S_i for each subtree of CST and subscribes it again. The subscribing process is finished till all primitive subscriptions arrived at their destinations and put into the PSMs of brokers.

```
Subscribing algorithm
Input: subscription S; composite subscription tree CST
Output: null
subscribe(S)
if S is a primitive subscription
  if(S.dest==OCId)
    PSM.put(S.SId, S)
  else
    forward S to the next hop
else
  for $PS_i \in S$
    if($PS_i$.dest==OCId)
      PSM.put($PS_i$.SId, $PS_i$)
    else HOPS.add($PS_i$.nexthop)
if HOPS.size==1
  forward S to the next hop
else
  CSM.put(S.SId,S)
  for $subtree_i \in$ CST
    $S_i$= CST.$subtree_i$
    $S_i$.src=OCId
    $S_i$.parentSId=S.Sid
    subscribe($S_i$)
```

Following the example in Fig. 8, broker N_1 received a composite subscription $S=((S_1 \text{ before } S_2)\&(-S_3)_T)$, and the tree structure of S is shown in Fig. 8(a). Broker N_1 first calculates the destinations (N_{1122}, N_{1121} and N_{111}) of all primitive subscriptions according to the OCId algorithm. Since the next hops of all primitive subscriptions are the same (N_{11}), the composite subscription is forwarded as a whole. Because the next hops are different, broker N_{11} first puts S into the CSM, then forwards $(-S_3)_T$ to broker N_{111}, and creates a new composite subscription (S_1 before S_2) and forwards it to broker N_{112}. Then broker N_{112} puts (S_1 before S_2) into the CSM, and forwards S_1 and S_2 to N_{1121} and N_{1122}, respectively.

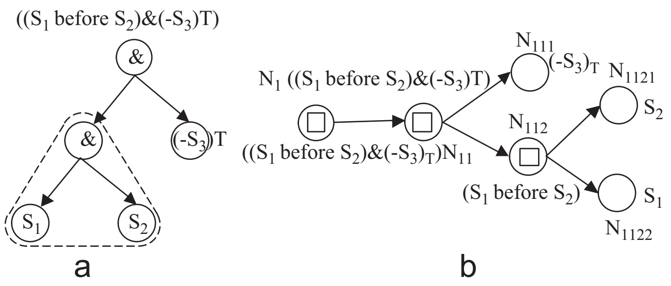

Fig. 8. Composite subscription tree and subscribing process.

If a subscriber wants to unsubscribe, it can send an unsubscription message. To ensure that the unsubscribing process is successful, an ACK message is sent if a broker successfully removes a subscription.

```
Unsubscribing algorithm
Input: subscription $S$; composite subscription tree CST
Output: null
unsubscribe($US$)
if ($US.dest$! = OCId)
  forward $US$ to the next hop
else
  if $US$ is a primitive unsubscription message
    PSM.remove($US$.SId)
    ACK.dest = US.src
    ACK.SId = $US$.SId
    ack(ACK)
  else
    $S$ = CSM.get($US$.SId)
    for $PS_i \in S$
      if($PS_i.dest$ = = OCId)
        PSM.remove($PS_i$.SId)
        ACK.$dest$ = $PS_i.src$
        ACK.SId = $PS_i$.SId
        ACK.parentSId = $PS_i$.parentSId
        ack(ACK)
    for subtree$_i \in$ CST
      $S_i$ = CST.subtree$_i$
      $US_i.src$ = OCId
      $US_i$.parentSId = $S$.SId
      $US_i$.SId = $S_i$.SId
      unsubscribe($US_i$)
ack(ACK)
if(ACK.$dest$! = OCId)
  forward ACK to the next hop
else
  if ACK.parentSId = = null
  sent ACK to the client
else
  $S$ = CSM.get(ACK.parentSId)
  S.count++;
  if(S.size = = S.count)
    CSM.remove($S$.SId)
    if($S$.parentSId = = null)
      sent a new ACK to the subscriber
    else
      ACK2.$dest$ = $S.src$
      ACK2.SId = $S$.SId
      ACK2.parentSId = $S$.parentSId
      ack(ACK2)
```

During the unsubscribing process, a broker receives an unsubscription message US. If the destination of US is not equal to OCId of the broker, the broker forwards it to the next hop. Otherwise, if US is a primitive unsubscription message, the broker removes the corresponding subscription from PSM and sends an ACK message back to the subscriber. If US is a composite unsubscription message, the broker gets the composite subscription S from CSM by SId, then for each primitive subscription PS_i of S, if the destination is equal to OCId, the broker removes PS_i from PSM and sends an ACK message back to the source broker where US is generated. Next, for each subtree of CST, the broker creates a new composite unsubscription US_i and does the unsubscribing process again. When a broker receives an ACK message, if the destination of ACK is not equal to OCId, the broker forwards it to the next hop. Otherwise, the ACK message arrives at its destination, if parentSId of ACK is null, the broker just sends the ACK to the subscriber. Else the broker gets the composite subscription S from CSM by the parentSId of ACK, and adds the value of count. If the $S.count$ is equal to the number of subtrees $S.size$, then the ACK messages from all subtrees of S are received, S is removed from CSM. If parentSId of S is null, an ACK message is send to the subscriber. Else, a new ACK message is created and forwarded.

Fig. 9 shows an example of the unsubscribing process. When broker N_1 receives a composite unsubscription message US, N_1 forwards US as a whole to broker N_{11}. Broker N_{11} gets S from the CSM, and creates new unsubscription messages for subtrees and forwards them to broker N_{112} and N_{111}. N_{112} does almost the same thing as N_{11}. Then N_{1121} and N_{1122} receive the unsubscription messages and remove primitive subscriptions from PSMs correspondingly, and send ACK messages to N_{11}. Finally, N_1 receives an ACK message and sends it to the subscriber.

4.4. Matching and notifying process

Because composite subscriptions are decomposed into primitive subscriptions during the routing process, so the matching process only needs to match primitive events with primitive subscriptions only.

```
Matching and notifying algorithm
Input: event E
Output: null
match($E$)
for $PS_i \in$ PSM
  if match($E$, $PS_i$)
    $N.E$ = $E$
    N.dest = PS$_i$.src
    $N$.SId = $PS_i$.SId
    $N$.parentSId = $PS_i$.parentSId
    notify($N$)
match($E$,$PS$)
if $E.c$ = = $PS.c$
  for p$_i \in$ PS.S$_{tr}$
    if $E.S_{tr}$ has $p_i$
      if $p_i$ is a datatype property
        if $PS.S_{tr}.p_i.o$ = = $E.S_{tr}.p_i.o$
          $b_i$ = true
```

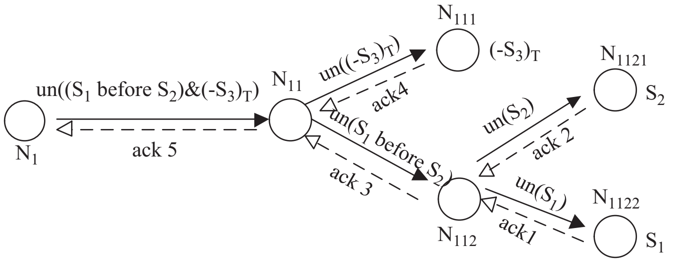

Fig. 9. Unsubscribing process.

```
        else if $p_i$ is an object property
            if match($E.S_{tr}.p_i.o$,$PS.S_{tr}.p_i.o$)
                $b_i$=true
    else
        for $p_k \in (E.S_{tr}$^semantic$(p_j))$
            if $p_k$ is a datatype property
                if $PS.S_{tr}.p_i.o==E.S_{tr}.p_k.o$
                    $b_i$=true
                    break
            else if $p_k$ is an object property
                if match($E.S_{tr}.p_i.o$,$PS.S_{tr}.p_k.o$)
                    $b_i$=true
                    break
            if $b_i$==false
                return false
        for $p_i \in$PS.F
            if $E.S_{tr}$ has $p_i$
                if ! mop($PS.F.p_i.o$, $E.S_{tr}.p_i.o$)
                    for $p_k \in (E.S_{tr}$^semantic$(p_j))$
                        if mop($PS.F.p_i.o$, $E.S_{tr}.p_k.o$)
                            $b_i$=true
                            break
                    if $b_i$==false
                        return false
    return true
    notify($N$)
    if ($N.dest$!=OCId)
        forward $N$ to the next hop
    else
        if $N$.parentSId==null
            $S$=PSM.get($N$.SId)
                send $N$ to the subscriber
        else
            EM=SEM.get($N$.parentSId)
            EQ=EM.get($N$.SId)
            EQ.add($N.E$)
            $S$=CSM.get($N$.parentSId)
            $CE$= createCE($S$)
            if($CE$!=null)
                $N.E$=$CE$
                N.dest=S.src
                $N$.SId=$S$.SId
                $N$.parentSId=$S$.parentSId
                notify($N$)
```

JTangCSPS supports semantic matching including inheritance and rule-based reasoning. Only the following three conditions are satisfied, we will say an event is semantically matched with the subscription: (1) the ontology class of the subscription is the same as that of the event; (2) for each triple of the subscription, there is at least one triple of the event whose subject and object nodes are semantically matched and linked by a semantically matched property arc and (3) for each filter of the subscription, there is at least one triple of the event whose subject node is matched, the value of object node satisfies the constraint of filter, and linked by a semantically matched property arc. The reason why JTangCSPS does not support the inheritance of class directly is, in that case, a subscription will be sent to the brokers that manage the children class of the subscription's class in the ontology class tree. This incurs significant network traffic overhead. The semantic(p) function of the matching process returns a set of properties, which are children of property p or can be reasoned by rules. The mop function returns the result of the mathematical operation as a boolean value. A notify message N has an event E, a destination $dest$, a SId and a parentSId. When an event matches a subscription, a notify message is created and routed to its destination.

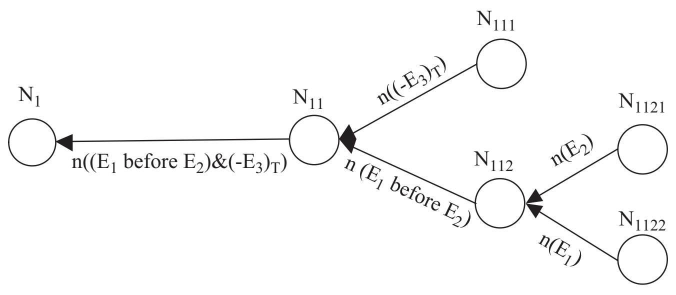

Fig. 10. Notification routing process.

To match the composite subscriptions, each broker has a subscription event map SEM and event map EM. The key of SEM is SId of composite subscription and the value is EM. The key of EM is SId of the child subscription and the value is an event queue EQ, which stores the events matched with the child subscription. When a broker receives a notify message N, if the destination of N is not equal to OCId, it forwards N to the next hop. Else, if the parentSId of N is null, which means N is sent to a subscriber who subscribes a primitive subscription, the broker gets the subscription S from PSM and sends N to the client directly. Else, if the parentSId of N is not null, the broker gets EM from SEM by parentSId of N, then gets EQ from EM by SId of N, and then adds $N.E$ to EQ. One of the benefits of distributed composite subscriptions management is, for each composite subscription, the broker only has to handle one logical or temporal operator. If the broker detects the events satisfied with the operators of composite subscription S by createCS function, it removes the events from EQs and creates a composite event CE, then puts it into a notification message N and routes N to the destination of parent subscription or the subscriber.

Fig. 10 shows an example of the notification routing process. Broker N_{112} receives notification messages from N_{1122} and N_{1121} and puts the events in EQs. When N_{112} detects events E_1 and E_2 satisfied with E_1 before E_2, it creates and forwards the composite event (E_1 before E_2) to N_{11} and removes them from EQs. At last, N_1 forwards the notification message with ((E_1 before E_2) and $(-E_3)_T$) to the subscriber.

5. Experiments

To measure the performance of the proposed system and to verify the concepts presented in this paper, we have developed a prototype of the system and have simulated the prototype with various parameter settings. We use PeerSim (Ulbrich et al., 2004), http://peersim.sourceforge.net/ simulator for performing experiments over Pastry overlays. All measurements took place on a standard PC installation with Linux libraries and a hardware configuration comprising 8 Intel Xeon CPU 2.00 GHz, 4 GB RAM. To simplify, the simulations are made over a static network that does not suffer from node join and neither node failures.

To simulate the semantic broker network, for each different network size, we build a full ontology class weighted tree and its height is h and the number of children is c. The weight of child class is the weight of the parent class plus a new weight. The new weight is generated according to Zipf (a=1) distribution. The weight of Broker is generated by Zipf (a=1) distribution too. The number of OCWT is about 3–4 times of the network size. The size of different broker network and the corresponding OCWT are shown in Table 1.

Table 1
Broker network and OCWT.

Number of broker	Nodes of OCWT	OCWT structure
1000	3616	$c=15$, $h=4$
3000	9331	$c=6$, $h=6$
5000	19,608	$c=7$, $h=6$
10,000	37,449	$c=8$, $h=6$
30,000	137,257	$c=7$, $h=7$
50,000	137,257	$c=7$, $h=7$
100,000	299,593	$c=8$, $h=7$

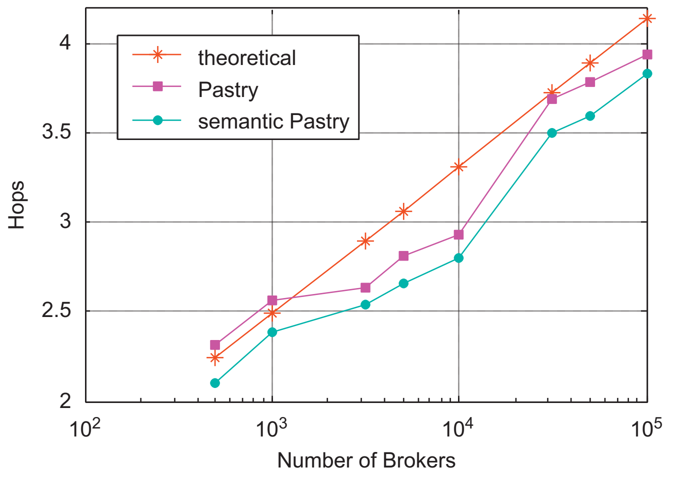

Fig. 11. Average number of hops.

5.1. Hops

Each broker random publishes primitive events and routes them to the destination node. We compare the average number of hops of events over Pastry overlay and the semantic Pastry overlay with ontology routing table. The result is made by varying the size of the broker network from $N=500$ to 10,000. The red line is the theoretical value of Pastry ($\log_2{}^bN$, $b=4$). Fig. 11 shows that the performance of Pastry is comparable with the theoretical expectations. The performance of semantic Pastry is better than that of Pastry. The ontology routing table adds some shortcuts in the broker network and decreases the average number of hops and so reduces the load of the network.

5.2. Routing delay

We route a composite subscription and split to the destinations of the primitive subscriptions. The routing delay of a composite subscription includes the network delay and the time to build the composite subscription tree and to split the composite subscription at each node. We define it as $T_{RD}=T_{ND}+T_C$ and $T_{ND}=\sum_{j=1}^{J} g(j)$ where $g(j)$ is the function of network delay and J is the maximum hops of the all primitive subscriptions. We define $f(i)$ as the function of time of building the composite subscription tree and splitting the composite subscription at a node where the number of primitive subscriptions of the composite subscription(NoPS) is i. So the maximum time will be $T_{C\max}=\sum_{i=2}^{I} f(i)$ when each node of tree has only a child node. The minimum time will be $T_{C\min}=f(T)$ when each primitive subscription has different next hops.

We compare a series of scenarios. To simplify, only operator conjunction "&" is used in composite subscription, and every primitive subscription only contains one type constraint. In the first scenario, each broker randomly creates 20 primitive subscriptions and registers to the broker network. In the other scenarios, instead of primitive subscriptions, each broker randomly creates $20/i$ composite subscriptions, each consisting of i primitive subscriptions where $i=2, 5, 10, 20$. Fig. 12(a) shows that the routing delay increases with the size of the network. Fig. 12(b) shows that the routing delay increases linearly with the number of hops of the composite subscription. Fig. 12(c) shows that the routing delay increases fast before NoPS reaches 5 and then increases slowly. Though the routing delay might be different for different performances of broker, different

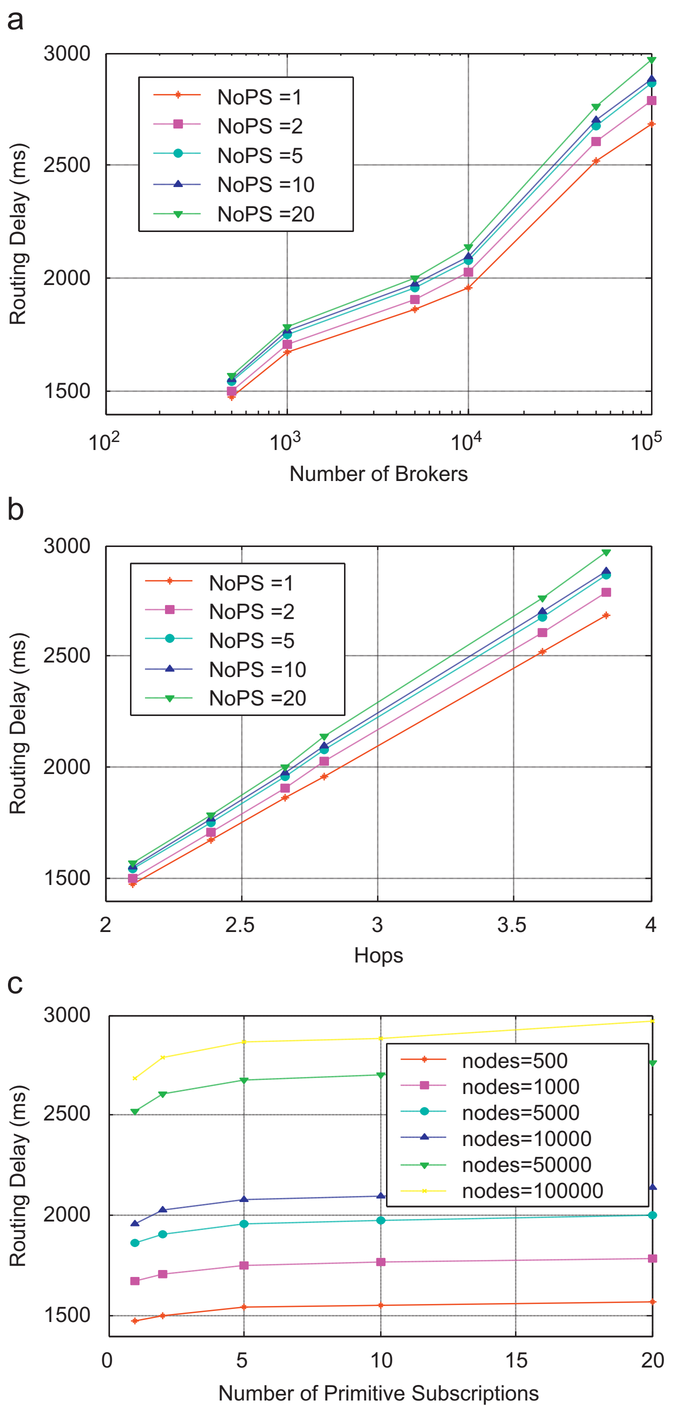

Fig. 12. Routing delay of composite subscription.

complexities of the composite subscription and different traffic of network, the results give us a qualitative analysis.

5.3. Composite subscription distribution

A composite subscription will be split into primitive or composite subscriptions while routing to the destinations. In different size of network, each broker randomly creates 100 composite subscriptions registered to them, each subscription consists of 5 primitive subscriptions. We count the number of composite subscriptions, which contain 2–4 primitive subscriptions and are generated during the routing process at each broker. Fig. 13 shows that, during the routing process, a composite subscription of 5 primitive subscriptions has a possibility of 50% to create a new composite subscription of 2 primitive subscriptions. It helps reduce an overall message traffic by collecting events as close as to the source nodes. However, due to Pastry's feature of hop, the possibility to generate a composite subscription with 3 or 4 primitive subscriptions is very small.

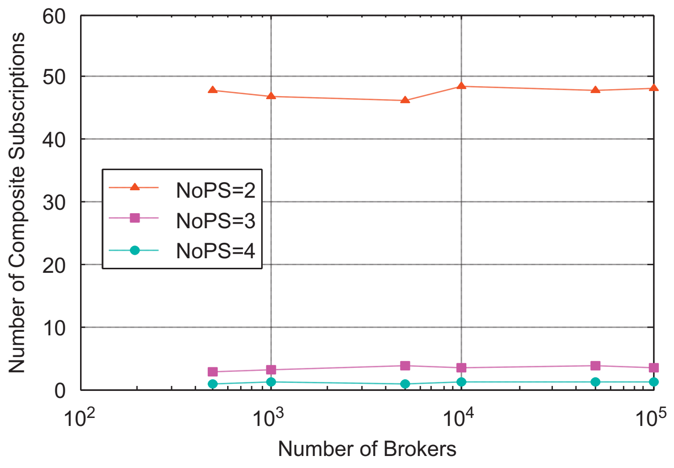

Fig. 13. Composite subscription distribution.

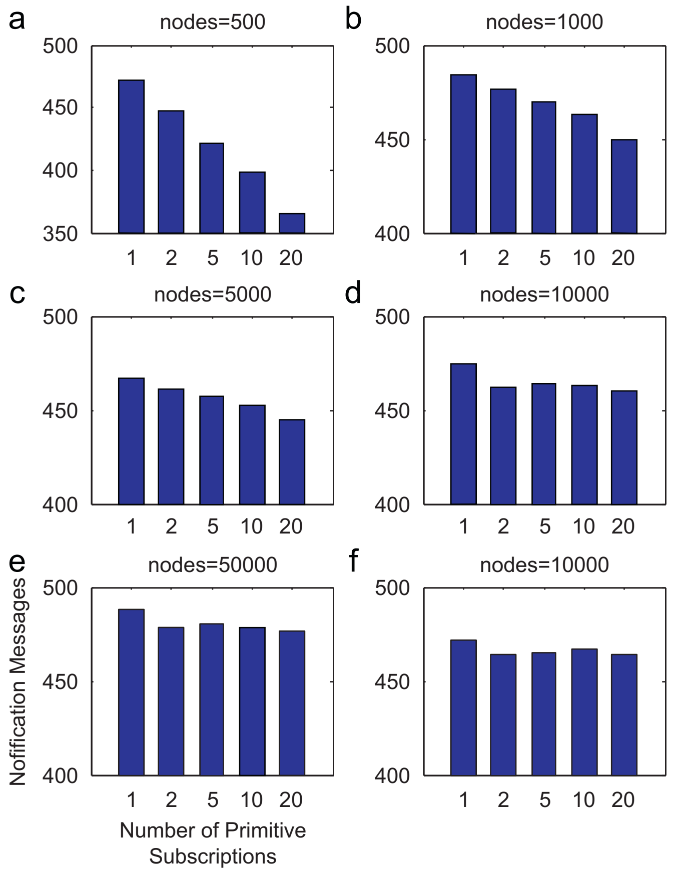

Fig. 14. Network traffic overhead of composite subscription.

5.4. Network traffic overhead

Detecting and composing composite events in the broker network reduce the message traffic received by clients. We use the same series of subscription scenarios of routing delay. After that, each broker publishes 50 events.

We count the number of notification messages passed at each broker in the different scenarios, which is shown in Fig. 14. The result shows that the number of notification messages reduces while the number of the primitive subscriptions of the composite subscription increases. The number of notification messages of smaller size of broker network reduced faster than that of bigger size of network. For example, when NoPS is 20 in network of 500 nodes, the number of notification messages reduces from 470 to 375 with a reduction of 20%.

6. Conclusion

In the paper, we present JTangCSPS, a composite and semantic publish/subscribe system over structured P2P network. We introduce ontology into the structured P2P network to provide semantic support, and define the weights of ontology classes and brokers and virtual subscription to map OCWT or COCWT to the broker network, achieve large-scale distributed computing and load balance. The ontology routing table maintains OCWT or COCWT and reduces the hops of messages over structured P2P network. Besides, the node backup strategy enhances the reliability of service. The paper also presents a composite event and subscription language to support the temporal and logical patterns of the distributed events. Distributed composite subscriptions management decomposes composite subscriptions and collects the primitive events from different brokers and aggregates them. It ensures that the events are not unnecessarily disseminated throughout the broker network. The experiments based on the Peersim simulator over the Pastry overlays show that the ontology routing table helps reduce the average number of hops, and the distributed composite subscriptions management significantly reduces the load on the network.

Acknowledgment

The work is supported by the National High-Tech Project of China (nos. 2011ZX01039-001-002 and 2009ZX01043-003-003). The authors would like to thank the reviewers for their helpful and constructive remarks.

References

Ahull, J.P., Lpez, P.G., Skarmeta, A.F.G., 2008. LightPS: lightweight content-based publish/subscribe for peer-to-peer systems. In: Proceedings of the International Conferance on Complex, Intelligent and Software Intensive Systems, Barcelona, Spain, pp. 342–347.

Allen, J.F., Ferguson, G., 1994. Actions and events in interval temporal logical. Logical and Computation 4, 531–579.

Baldoni, R., Marchetti, C., Virgillito, A., Vitenberg, R., 2005. Content-based publish/subscribe over structured overlay networks. In: Proceedings of the International Conferance On Distributed Computing Systems (ICDCS '05), pp. 437–446.

Buchmann, A., Bornhövd, C., Cilia, M., Fiege, L., Gärtner, F., Liebig, C., Meixner, M., Mühl, A., 2004. DREAM: distributed reliable event-based application management. In: Levene, M., Poulovassilis, A. (Eds.), Web Dynamics. Springer-Verlag (pp. 319–352).

Chirita, P.A., Idreos, S., Koubarakis, M., Nejdl, W., 2004. Publish/subscribe for RDF-based P2P networks. In: Proceedings of the 1st European Semantic Web Symposium, pp. 182–197.

Courtenage, S., 2002. Specifying and detecting composite events in content-based publish/subscribe systems. In: Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, pp. 602–607.

Courtenage, S., Williams, S., 2006. The design and implementation of a P2P-based composite event notification system. In: Proceedings of the 20th International Conference on Advanced Information Networking and Applications, pp. 701–706.

Crespo, A., Garcia-Molina, H., 2003. Semantic Overlay Networks for P2P Systems. Technical Report. Stanford University, ⟨http://ilpubs.stanford.edu:8090/627/⟩.

Demers, A.J., Gehrke, J., Hong, M.S., Riedewald, M., White, W.M., 2006. Towards expressive publish/subscribe systems In: Proceedings of the Advances in Database Technology (EDBT), pp. 627–644.

Demers, A.J., Gehrke, J., Panda, B., Riedewald, M., Sharma, V., White, W.M., 2007. Cayuga: a general purpose event monitoring system. In: Proceedings of the CIDR Conference, pp. 412–422.

Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.-M., 2003. The Many Faces of Publish/Subscribe. ACM Computing Surveys 35 (2), 114–131.

Gang, X., Wei, X., Tao, H., 2005. Extending OBDD graphs for composite event matching in content-based Publish/subscribe systems. In: Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC), Lille, France, pp. 290–298.

http://peersim.sourceforge.net/.

Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21 (7), 558–565.

Li, G.L., Jacobsen, H.A., 2005. Composite subscriptions in content-based publish/subscribe systems. In: Proceedings of the 6th ACM/IFIP/USENIX International Middleware Conference 2005, pp. 249–269.

Löser, A., Naumann, F., Siberski, W., Nejdl, W., Thaden, U., 2003. Semantic overlay clusters within super-peer networks. In: Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 03), pp. 33–47.

McGuinness, D.L., Fikes, R., Stein, L.A., Hendler, J., 2002. DAML-ONT: an ontology language for the semantic web. IEEE Intelligent Systems, 1541–1672.

Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., Lser, A., 2003. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In: Proceedings of the 12th World Wide Web Conference, pp. 536–543.

Petrovic, M., Burcea, I., Jacobsen, H.A., 2003. S-ToPSS: semantic Toronto publish/subscribe system. In: Proceedings of the 29th International Conference on Very Large Data Bases, pp. 1101–1104.

Petrovic, M., Liu, H., Jacobsen, H.A., 2005. G-ToPSS: fast filtering of graph-based metadata. In: Proceedings of the 14th International Conference on World Wide Web, pp. 539–547.

Pietzuch, P.R., Bacon, J.M., 2002. Hermes: a distributed event-based middleware architecture. In: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCSW '02), pp. 611–618.

Pietzuch, P.R., Shand, B., Bacon, J., 2003. A framework for event composition in distributed systems. In: Proceedings of the ACM/IFIP/UNSENIX International Middleware Conference, pp. 62–82.

Pietzuch, P.R., Shand, B., Bacon, J., 2004. Composite event detection as a generic middleware extension. IEEE Network 18, 44–55.

Pujol-Ahullo, J., Garcia-Lopez, P., Gomez-Skarmeta, A.F., 2009. Towards a lightweight content-based publish/subscribe services for peer-to-peer systems. International Journal of Grid and Utility Computing 1 (3), 239–251.

Qian, J.F., Yin, J.W., Dong, J.X., 2011. Distributed management of composite subscriptions of semantic pub/sub system. In: Proceedings of the Third International Conference on Computer and Network Technology, vol. 4, pp. 472–476.

Raftopoulou, P., Petrakis, E.G.M., 2008. iCluster: a self-organising overlay network for P2P information retrieval. In: Proceeding of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval (ECIR), pp. 65–76.

Rowstron, A., Druschel, P., 2001. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proceedings of the 2th IFIP/ACM International Middleware Conference 2001, pp. 329–350.

Schmitz, C., 2004. Self-organization of a small world by topic. In: Proceedings of the 1st International Workshop on Peer-to-Peer Knowledge Management (P2PKM).

Ulbrich, A., Mühl, G., Weis, T., Geihs, K., 2004. Programming abstractions for content-based publish/subscribe in object-oriented languages. In: Proceedings of the CoopIS/DOA/ODBASE (2), pp. 1538–1557.

Wang, Jin-Ling, 2005. Research on key technologies in internet-scale publish/subscribe systems. Ph.D. Thesis. Institute of Software, The Chinese Academy of Sciences.

Yoneki, Eiko, Bacon, Jean, 2005. Unified semantics for event correlation over time and space in hybrid network environments. In: Proceedings of the IFIP International Conference on Cooperative Information Systems (CoopIS), pp. 366–384.