QTor: a flexible Publish/Subscribe peer-to-peer organization based on query rewriting

Sébastien Dufromentel, Sylvie Cazalens
François Lesueur, and Philippe Lamarre

Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205
F-69621, France
<firstname.lastname@liris.cnrs.fr>

Abstract. Peer-to-peer publish/subscribe architectures are an interesting support for scalable distributed data stream applications. Most approaches, often based on brokers, have a static organization which is not much adaptive to different configurations of the participants' capacities. We present *QTor* (*Query Torrent*) a generic organization that enables dynamic adaptation providing a continuum from centralized to fully decentralized solutions. Based on query rewriting and equivalence, *QTor* proposes a definition of *communities* and their relations that decouples the logical and physical aspects of the problem, while efficiently reducing organizational and functional costs.

1 Introduction

Peer-to-peer publish/subscribe organizations are an essential support for scalable distributed data stream applications. A lot of them [5,8] rely on brokers to benefit from efficient local processing optimizations. More recent propositions [6,12] are fully distributed, asking the users to contribute by sharing their results, which requires to take care of their limited resources.

Among several possible overlays to organize the system, query-oriented organizations take advantage of already computed results [4,2,6]. In this field, an approach based on query rewriting [9,1] provides the most general solution that is both generic and language independent and that eases the decoupling of logical and physical layers. In addition, we consider the case of popular queries which, even if their expressions differ, are *equivalent*, which means they give the same results when executed on the same dataset.

We present QTor, an organization driven by the queries in which participants are grouped into *communities*, regarding query equivalence relation. Those communities are independent from each other and autonomous for their local organization. Connections between communities come from query rewritings combined with multiple criteria (computing and networking costs, latency...). This results in a system with low organizational and functional costs that is adaptive to the incoming participants' capacities while preserving a low latency.

In Section 2, we present some preliminaries and formally define the problem. The main concepts of our approach and the resulting system are described in Section 3. Section 4 discusses QTor flexibility. Experimental results of Section 5 show its good performances, and draws a comparison with two other approaches. Finally, Section 6 presents related work while Section 7 concludes and exposes some perspectives.

2 Background and problem statement

In this section, we formally define the problem as finding a system on which, for any submitted query, a rewriting graph can be mapped, while complying with the participants' limitations.

2.1 System organization based on query rewriting

A distributed system is composed of software *participants* which originate some streams, submit users' queries, and bring the system some resources (memory, storage and computational capacities).

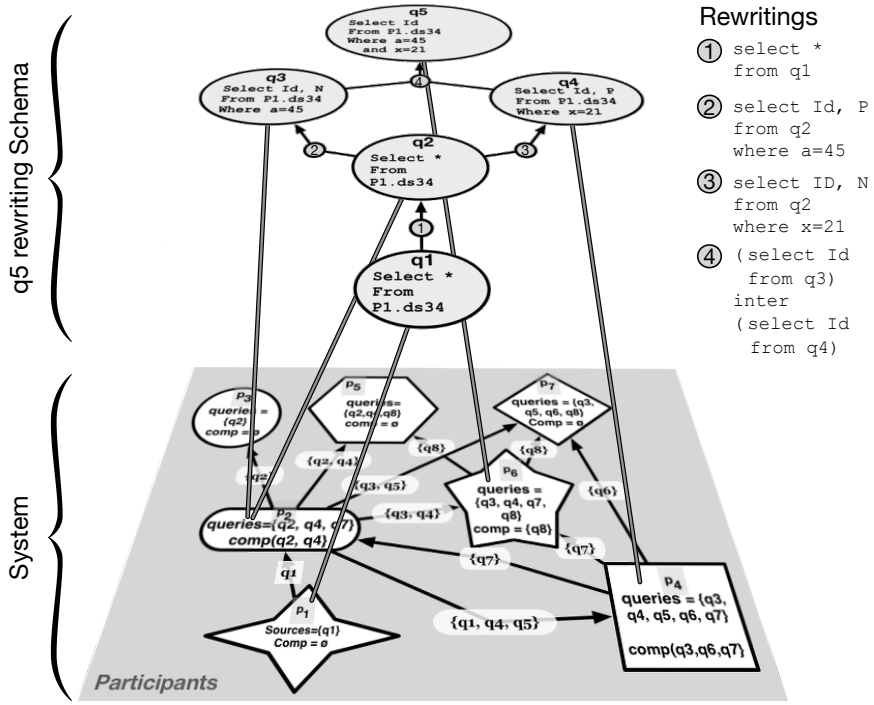
Definition 1 (System). *A* system *is a labelled graph* $\langle P, PA \rangle$ *such that:*
- P *is a set of participants. For each participant* $p \in P$, $p.sources$ *is the set of queries representing the data streams* p *originates and* $p.queries$ *is the set of queries it submits to the system.*
- PA *is a set of triples* $\langle p_i, p_j, Q_{i,j} \rangle$ *with* $(p_i, p_j) \in P^2$ *and with* $Q_{i,j}$ *non empty finite set of queries representing the data streams* p_j *gets from* p_i*.*

The bottom part of Figure 1 is an example of such system.

In order to organize such a system, relations between queries have to be studied. Two queries q_1 and q_2 are equivalent, noted $q_1 \equiv q_2$, if and only if they provide the same results regardless of the sources' data streams contents. Rewriting query q means finding a query q' equivalent to q but expressed over a set of queries q_i (representing data streams or queries) different from the one used to express q. This is noted $q' = (q \longleftarrow \{q_1, q_2, \ldots, q_n\})$, or simply $q \longleftarrow \{q_1, q_2, \ldots, q_n\}$, when q' is of no need.

Definition 2 (Rewriting schema). *A rewriting schema of a query* q_r*, is a cycle free, totally connected graph* $\langle Q, E \rangle$*, with* Q *a set of queries and* E *a set of arcs, characterized by* $q_r \in Q$ *with:*
- $\forall \{q, q_1, q_2 \ldots q_n\} \in Q^{n+1}$*, if* $(\{q_1, q_2 \ldots q_n\}, q) \in E$ *then there exists a rewriting* $q \longleftarrow \{q_1, q_2, \ldots, q_n\}$*, and*
- *the in-degree of each node is at most one, and*
- q_r *is the only node with an out-degree equal to zero.*

Despite the number of possible rewritings between queries, a rewriting schema contains at most one possible rewriting for each query and avoids cycles, as the configuration graph used in [6]. An example of rewriting schema is illustrated on the upper part of Figure 1.



System: Participant $p1$ is the unique source participant that originates a data stream ($p1.ds34$, represented by $q1$). Participant $p2$ submits queries $q2$, $q4$ and $q7$ to the system. It gets $q1$ from $p1$ and $q7$ from $p4$. It computes $q2$ and $q4$. It serves $q1$ to $p4$; $q2$ to $p3$ and $p5$; $q3$ to $p6$ and $p7$; $q4$ to $p4$, $p5$ and $p6$; $q5$ to $p4$ and $p7$. And so on for other participants.

Rewriting schema for $q5$: $q2$ (`select * from p1.ds34`) is rewritten from $q1$ (`select * from p1.ds34`) using identity (noted rewriting ① on the figure) ; $q3$ and $q4$, are rewritten using $q2$ with a selection and a projection (`select Id, P from q2 where x=21`) ; $q5$ is rewritten as the intersection of $q3$ and $q4$.

Fig. 1. The system implements a rewriting schema for $q5$.

Definition 3 (Well founded system). *A system $\langle P, PA\rangle$ is well founded if and only if for any query q issued by any participant p there exists a rewriting schema $\langle q_r, Q_r, E_r\rangle$ such that:*

- $q_r = q$ *(syntactic equality), and*
- *there is a mapping m from Q_r to P such that*
  - $m(q_r) = p$, *and*
  - $\forall(\{q_1, q_2 \dots q_n\}, q_g) \in E_r$, $\forall q_i \in \{q_1, q_2 \dots q_n\}$, $(m(q_i) = m(q_g)) \vee (\langle m(q_i), m(q_g), Q_{i,g}\rangle \in PA \wedge q_i \in Q_{i,g})$, *and*
  - $\forall q_i \in Q_r$, *if q_i is a leaf then $q_i \in m(q_i).sources$.*

The use of rewriting schemas ensures each participant can compute the results it is interested in using the streams it gets. Furthermore, because the rewriting schema is acyclic, the system does not embed any cycle. This is important to avoid loops, which make participants waiting for each others' results.

2.2 Problem statement

To further define the problem it is necessary to pay attention to real world constraints. *Physical constraints:* the participants have limited computational and network capacities. Overload of each capacity has to be avoided. Moreover, the solution should consume as few resources as possible considering not only its functional cost, but also organizational costs (setting up and maintenance costs). *Social constraints:* the solution has to comply with each participant's resource management policy that specifies to which uses the resources that it provides are dedicated. Last but not least, in many applications, latency is of prime importance for users' welfare.

The problem is to find a *well founded system*, which is i) consistent with participants capacities, ii) compliant with their resource management policies, iii) as efficient as possible with respect to resources consumption (computing, network, energy), and iv) providing results to participants with a latency as small as possible.

3 QTor system

We present *QTor* (*Query Torrent*), a generic well founded system which addresses the problem. We focus on its organization principles, which ensure the system flexibility avoiding details of the participants' inner management. Our approach is based on the notion of community, which is in charge of the computation of a set of equivalent queries and of the resulting data stream. A graph of those communities provides the overall organization.

3.1 Graph of communities

We define a ***computing unit*** as a software component which is created by a participant p to declare some interest in a query q, formally noted $u = \langle q, p\rangle$.

A participant creates a unit for each expressed or computed query. In addition, each unit requires sufficient network ressources to get its data streams in and to transmit its data stream output to at least another unit.

Definition 4 (Community). *A* community C *is a couple* $C = \langle q, U \rangle$ *where* q *is a query and* U *is a set of computing units such that* $\forall u \in U$, $u.q \equiv q$.

Notice that we do not require all units with equivalent queries to be part of the same community, due to pragmatic reasons. Indeed, all equivalences are not always provable in a reasonable time.

As a community C is defined by a query $C.q$, the semantic overlay can be structured taking advantage of rewriting technics.

Definition 5 (Graph of Communities). *A* Graph of Communities *is a cycle free hyper-graph* $\phi = \langle \mathscr{C}, RR \rangle$, *with*

- $\mathscr{C}$ *is a finite set of communities where* $\forall (C_i, C_j) \in \mathscr{C}^2$, *if* $i \neq j$ *then* $C_i \cap C_j = \emptyset$.
- RR *is a set of hyper-arcs such that:*
  - *if* $(\{C_1, C_2 \ldots, C_n\}, C) \in RR$ *then there exists a rewriting* $C.q \longleftarrow \{C_1.q, C_2.q, \ldots, C_n.q\}$, *and*
  - *the in-degree of any source-community is equal to zero ; the in-degree of any other community is equal to one.*

As each community is dedicated to a query (or a set of equivalent queries), the graph of communities describes and implements a rewriting schema.

The use of communities has three major effects. First, there are often much less communities than units, due to the popularity of queries. This reduces the graph size. Second, it avoids wasting time and energy to compute several times the same rewritings for each equivalent query (and it also avoids cycles between those equivalent queries). Third, it splits the physical organization problem into several smaller, independent tasks, while enabling units to collaborate when they compute complex queries which would be out of reach of a single participant.

Materializing this theoretical model into a functional system requires to answer some concrete questions like: “how does a community get the data streams it needs?” ; “how to disseminate the data stream corresponding to the result of a query?” ; and “how to compute a query within a community?”. Without loss of generality, we simplify this latter point by assuming that the computation is ensured by a single unit with enough resources. Indeed, introducing distributed computing within a community is an orthogonal problem which does not impact the other points.

3.2 Building the whole system

Reorganizing the whole system at each insertion, without considering the existing system, may be too expensive to implement. A periodic reorganization as in Delta[6] is possible, but it is difficult to define this periodicity and how to handle insertion of new participants during the period. This is why we choose

to rely on an incremental approach. For example, when a participant expresses a query, it searches for an existing community working on an equivalent query, through a tracker that knows all the communities and their organization. If one is found, the participant creates a unit which joins it. Otherwise, a new community is created: a cost model, considering processing, network loads and latency is used to select the best rewriting schema among those computed using the existing communities already organized according to rewriting schemas. Then, the new community has to be connected with the others. Notice that, contrary to a data integration problem, in this case, it is not necessary to compute all the possible rewritings [9].

3.3 Participants' spreading over communities

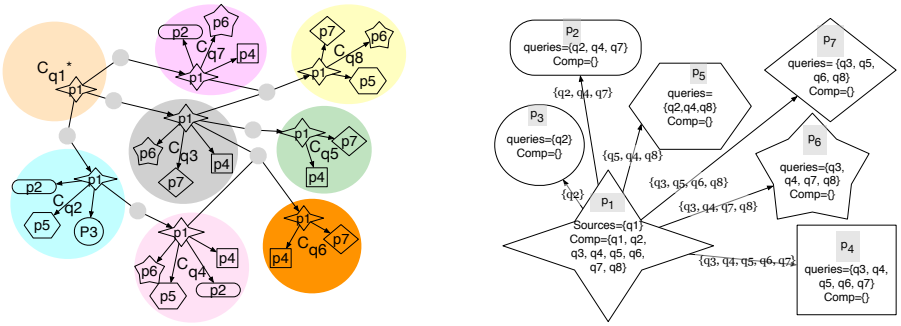
Given a community C, associated with query q and taking its data stream input from another community C', we define two mechanisms enabling a participant to create a new unit to contribute to an additional community. Notice that these mechanisms keep the communities disjoint and that they can be generalized to a rewriting involving several parent communities.

SPC: participant's Spreading to a Parent Community. A participant p_1 having a unit u_1 in community C creates and introduces a new unit u'_1 into community C' which enables p_1 to obtain the data stream from C'. From this, assuming u_1 (i.e. p_1) is powerful enough, it computes the rewriting of q defined by the rewriting schema and spreads the resulting data stream to other units of community C.

SCC: participant's Spreading to a Child Community. A participant p_2 having a unit u'_2 in the community C' creates and introduces a new unit u_2 into community C. u_2 has to be powerful enough to compute the rewriting query of q defined by the rewriting schema. Then, it spreads the resulting data stream to other units of community C.

In both cases, the new unit participates to its community tasks as any unit does. Depending on the situation, one solution may be better than the other. For example, due to a high selectivity of query q, the data stream of community C can have a lower throughput than the one of C'. If in addition, the rewriting of query q is easy to compute, the SCC mechanism becomes particularly interesting.

Choosing among these two mechanisms has to be done when a participant joins the system with a query that requires to create a new community. This choice is questioned at the departure of units concerned by one of these mechanisms or at the arrival of a new unit that is more powerful than those implied by the mechanism. This choice depends on specific situations, so this decision is left to participants, who will decide to use one or the other mechanism. Notice that communication between two communities is thus ensured by a same participant, involving a unit in each of them. Hence, the stream exchange is implemented without consuming ressources theoretically allocated to the community C', which would create dependencies between communities. As a consequence, each community is autonomous to organize itself.



(a) Graph of communities. (b) Corresponding system.

Fig. 2. Powerful data stream producer that accepts to compute for others.

3.4 Inner organization of a community

Due to the previous choices, each community can organize itself autonomously. The SCC and SPC mechanisms and the assumption that a query computation is done by a single unit define the way both data acquisition and query computation are ensured. The computing unit is the one belonging to the participant in charge of the incoming communication mechanism.

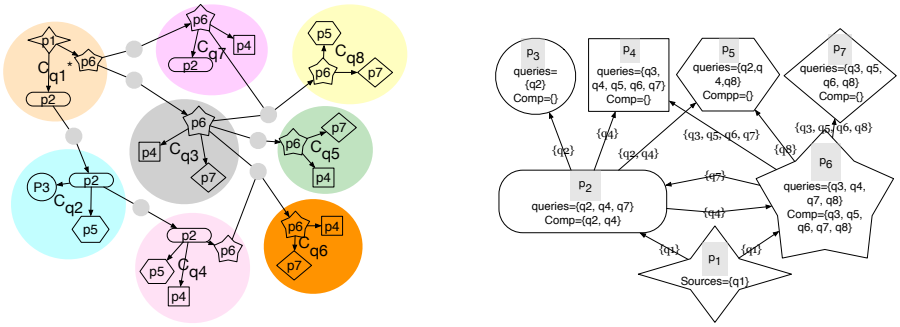
Results dissemination within a community is formalized as a diffusion tree with minimum latency. Under the simplifying assumption that the network is homogeneous we search for a minimal depth spanning tree rooted by the computing unit.

4 QTor flexibility

An interesting property of our proposal is its ability to adapt to different situations. We illustrate this point with two scenarios involving several subscribers and a single source $p1$ that provides a data stream represented by query $q1$ (same elements as in Figure 1). In our figures, a participant is represented by a specific form (e.g. circle, star...) which is also used for its units, with a smaller size.

In Figure 2, $p1$ is powerful enough to compute all queries and to disseminate results. By using several times the SCC mechanism, it creates a unit in each community leading to Figure 2(a). Thus $p1$ computes all queries and broadcasts results. The other participants get their results directly from the data producer without any effort. Figure 2(b) describes the resulting system which turns out to be a classical publish/subscribe system.

In Figure 3, the producer can send its data stream to many units but has no computing capability. Hopefully, some participants are powerful enough: $p2$ and $p6$ use SPC to get data stream from $p1$. Participant $p6$ uses also SCC to create units in community C_{q5}. Participant $p2$ already has units in C_{q2} and C_{q4} so using SPC or SCC does not change anything. As shown on figure 3(b) participants $p2$ and $p6$ bear all the queries computation and most of the diffusion.



(a) Graph of communities. (b) Corresponding system.

As in Figure 1, $p1$ originates the data stream represented $q1$

Fig. 3. A data stream producer with no computation capacity, with altruist consumers.

The limitation of the number of connections provided by data sources is a problem that dissemination systems are facing. A benefit of the SPC mechanism is to enable participants to contribute to the diffusion of the source streams, by creating units in the source community (as $p2$ and $p6$ do in the previous scenario).

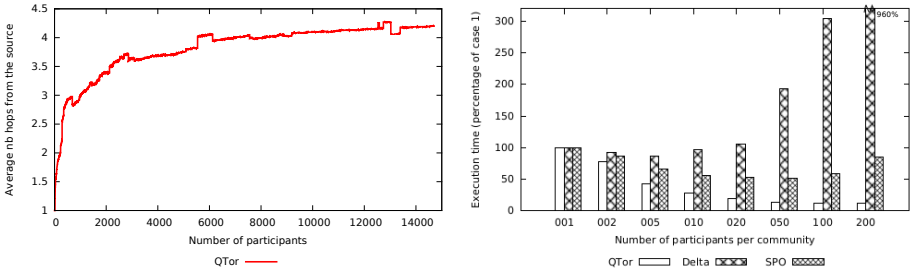
Based on fairly simple concepts such as graph of communities and the creation of units to provide data streams to communities, the formalism captures systems previously considered as very different. As a consequence, our approach is very flexible providing a continuum from centralized to fully decentralized solutions.

5 Experiments

We perform several experimentations on a java simulator using PeerSim[7]. We focus on the efficiency and the adaptability of the QTor organization. Thus without loss of generality, we consider participants expressing boolean queries (using both conjunctive and disjunctive operators) over a single data source, which publishes some randomly generated tuples. Participants capacities follow a Poisson law around 30. We use [4] and [6] as baselines.

Semantic peer-to-peer overlays [4] (called later "SPO") is a containment-based approach that places participants in a common spanning tree. Query equivalence is used to guide the organization (participants sharing equivalent queries are linked together, as this is the best way to reduce costs), but without any explicit notion of community: placements are computed considering the whole graph of participants. The original approach does not take care of the participants' capacities, considering a single node can send data to all the others. We developed an algorithm to fix this problem, and did not compare the aspects that were mainly related to this algorithm (Figure 4 (a), Figure 5, Figure 7 (b)).

Delta [6] is a recent proposition in which each participant is used as a view in a rewriting system. The system organization is obtained in five steps: an embedding graph of queries is computed, from which cycles are removed using a generic algorithm. It is used to build a rewriting graph. An ILP solver is then called to find a minimal cost solution under capacity constraints. Then, a last algorithm, called LOGA, reduces the distance between participants and the source to decrease latency (which is a non linear problem) avoiding to connect them directly to the source. However, it may choose rewritings that need more processing than the one chosen by the ILP solver.



(a) Latency evolution in QTor, size of communities following a Zipf distribution

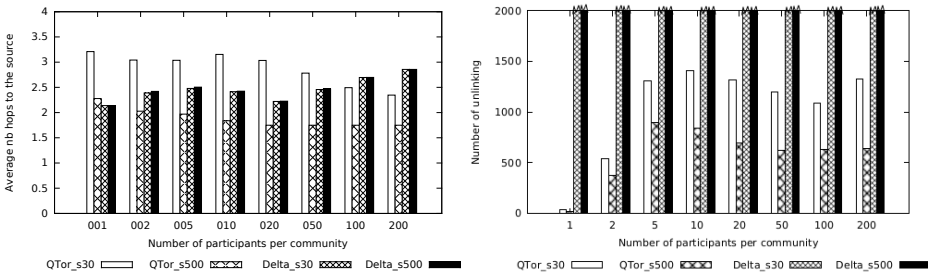
(b) Time spent to organize all the 2.000 participants, same size for all communities

Fig. 4. Scalability of the organizations

Figure 4 (a) illustrates QTor scalability. We consider up to 15.000 participants with popular queries following a Zipf distribution. Neither Delta nor SPO are shown here. Indeed, this kind of setup results in huge communities grouping more than 500 participants. Delta (especially the LOGA part) becomes very slow with such communities and it is too long to obtain results. SPO is not relevant.

In the following, to conduct comparisons, we consider smaller systems with up to 2.000 participants and a variable query popularity, i.e. community size.

In Figure 5 we evaluate the ability of the systems to adapt to different source capacities. We consider two cases: One source with capacities similar to other participants' with 30 output connexions. Another one with 500 output connexions. Figure 5 (a) shows the average depth of participants in the final system which is related to the latency of obtaining results for participants. Delta does not distinguish between those two situations. Indeed, participants are linked to the source if and only if they have no other possibility. This means that Delta may overload or under use the source. In QTor, the source is never overloaded (other participants take extra charge using SPC mechanism) and, as the other participants, a source can deploy units and participate to many communities (up to its capacity limit, using SCC mechanism) which is a way to reduce latency. In Figure 5 (b) we evaluate the stability of the organization and so the cost of the maintenance of the system. In Delta, it is recommended to run the reorganization

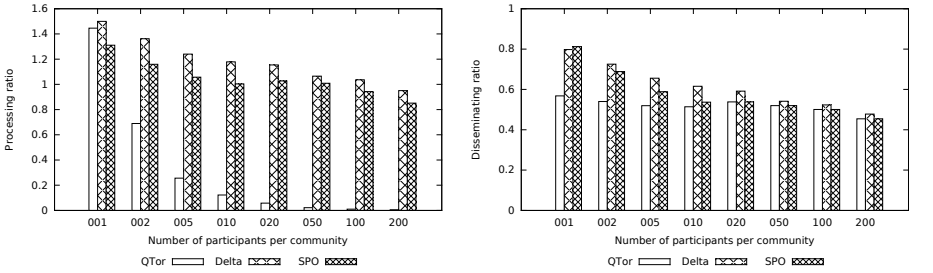


(a) Average depth (b) Number of reconnections

2.000 participants, variable source capacity (30 or 500), same size for all communities.

Fig. 5. System organisation

algorithm periodically. In its current form, Delta does not consider the existing system, and it may impact almost all the system graph. So, even with a low frequency, participants' re-connexions are more numerous than in our solution, and a powerful source is of no help. In QTor, the insertion of a new participant may lead to some community reorganizations, but its impacts are circumscribed to a part of the system graph. Furthermore, QTor takes advantage of a powerful source to limit the organization cost.



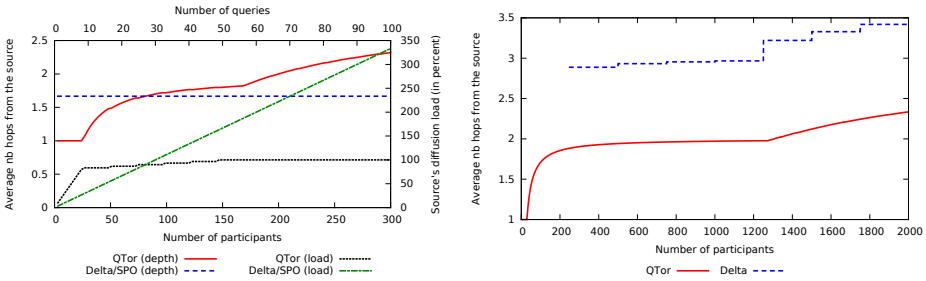
(a) Comparison of processing ratio (b) Comparison of disseminating ratio

2.000 participants, same size for all communities.

Fig. 6. Evaluation of functional costs

In Figure 6 we evaluate the functional costs associated to the organization (which is independent from the participants' fan-out limitation). (a) shows the processing ratio (which means, the average number of analysis performed per produced tuple), while (b) shows the disseminating ratio (the average number of tuples delivered by a participant per number of tuples produced by the source). SPO has good results, but their limitation to containment relations (which means one view per rewriting) makes them loose some opportunities. Delta's results in

low redundancy are higher than QTor's one for the same reasons its latency is lower: algorithm LOGA makes the functional costs increase. In QTor, communities avoid this fact.



(a) Independant queries, 3 participants per community

(b) Single community

Source capacity: 30 in both cases.

Fig. 7. Limit cases for the organization

In Figure 7 we study two limit cases: i) the queries of the communities are separate, i.e. they cannot be rewritten one for the others, and ii) all the participants are grouped into a sigle community, i.e. they all query for same results. In the first case, QTor is the only proposition providing a mechanism that avoids source's overwhelming. In the second case, the QTor organization has an efficient inner organization, while Delta's one is not really adapted.

6 Related work

Some distributed Publish/Subscribe systems, with brokers such as SemCast [8], or deployed over all the participants in a full peer-to-peer organization such as DPS [2], deploy complex organizations having several diffusion tree for the different predicates or several indexing levels. DHTrie [12] uses an extended Distributed Hashtable to register subscriptions, which, as often, highly depends on the chosen structure (query language, data schema). This gives organizations that are hard to adapt, while the use of rewritings allows a generic organization that can efficiently be adapted.

Some other distributed systems, like CDNs [10] or Multicast-inspired systems (for instance, SplitStream [3]), are based on the possibility for some participants to obtain all the data produced by the source, and to re-send them unchanged to the others. Such approaches lead to huge, redundant processing, as each participant has to work by its own, but may be used to design efficient diffusion scheme inside a QTor community.

A lot of propositions deal with local multi-query optimizations. For instance, in NiagaraCQ [5], grouping the query by their "signatures" based on the physical

operators allow to limit the I/O. In RoSeS [11], a local query graph is optimized by factorization. Those kind of propositions alone do not cope with huge number of participants, which require a network-wide organization, but may, in a QTor system, efficiently be used to decrease the processing load of participants that compute for several communities.

To the best of our knowledge, the closest propositions to ours are [4] and Delta [6]. SPO [4] is a very dynamic and incremental solution, but it is limited to containment and does not address the participants' capacity limitations. Delta [6] is based on query rewriting but proposes a global, non incremental, system reorganization with higher organizational and functional costs without decisive advantage on latency. Furthermore, it is not adapted at all to popular queries. These systems don't take full advantage of powerful sources/participants and may overwhelm sources which is a problem our solution avoids. Finally, the notion of query-based community introduces an intermediate abstraction which favors scalability in presence of popular queries.

7 Conclusion and perspectives

This paper shows that the QTor approach provides an efficient organization, taking care of several aspects to result in a reliable system: capacity limitations are never overwhelmed and participants having high capacity can efficiently help the others. The use of communities enables to highly reduce redundant works in case of popular queries, while simplifying the organization.

Those results are even more interesting considering there still are possible improvements: in this paper, collaboration inside a community is limited to sharing results. When the chosen rewritten query is still unreachable for a single participant, our model enables distributed processing, shared by participants having similar interests. Moreover, it may be possible to create communities that do not correspond to any expressed queries, but enable to share common works for several existing queries, as in the way locally used in RoSeS[11].

Decoupling logical and physical aspects allows to efficiently take care of both of them, without concurrent objectives, in a flexible, reliable organization. Some aspects still have to be considered, but the already obtained effects allow to consider that the QTor approach promises interesting surroundings.

Acknowledgements This work has been partially funded by the French ANR SocioPlug project under grant No. ANR-13-INFR-0003.

References

1. Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 254–263, New York, NY, USA, 1998. ACM.

2. E. Anceaume, M. Gradinariu, A.K. Datta, G. Simon, and A. Virgillito. A semantic overlay for self- peer-to-peer publish/subscribe. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 22–22, 2006.
3. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast in cooperative. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM Request Permissions, December 2003.
4. Raphaël Chand and Pascal Felber. Semantic peer-to-peer overlays for publish/subscribe networks. In JoséC. Cunha and PedroD. Medeiros, editors, *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 1194–1204. Springer Berlin Heidelberg, 2005.
5. Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for internet databases. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 379–390, New York, NY, USA, 2000. ACM.
6. Konstantinos Karanasos, Asterios Katsifodimos, and Ioana Manolescu. Delta: Scalable data dissemination under capacity constraints. *PVLDB*, 7(4):217–228, 2013.
7. Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, September 2009.
8. O. Papaemmanouil and U. Cetintemel. Semcast: Semantic multicast for content-based data dissemination. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 242–253, 2005.
9. Rachel Pottinger and Alon Halevy. Minicon: A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2-3):182–198, September 2001.
10. Iradj Ouveysib Tolga Bektasa, Osman Oguza. Designing cost-effective content distribution networks. In *Computers and Operations Research*, October 2005.
11. Jordi Creus Tomàs, Bernd Amann, Nicolas Travers, and Dan Vodislav. RoSeS: a continuous content-based query engine for RSS feeds. In *DEXA'11: Proceedings of the 22nd international conference on Database and expert systems applications*. Springer-Verlag, August 2011.
12. Christos Tryfonopoulos, Stratos Idreos, Manolis Koubarakis, and Paraskevi Raftopoulou. Distributed large-scale information filtering. In Abdelkader Hameurlain, Josef Küng, and Roland Wagner, editors, *Transactions on Large-Scale Data- and Knowledge-Centered Systems XIII*, volume 8420 of *Lecture Notes in Computer Science*, pages 91–122. Springer Berlin Heidelberg, 2014.