Improving Validity of Query Answering in Dynamic Systems

Roberto Baldoni Silvia Bonomi Adriano Cerocchi Leonardo Querzoni
Sapienza Università di Roma, Via Ariosto 25, 00185 Roma, Italy
{baldoni,bonomi,cerocchi,querzoni}@dis.uniroma1.it

ABSTRACT

Let us consider a large scale distributed system and a query executed on top of it where every process has to contribute to the result. Informally, a query satisfies the interval validity property if its result has been calculated by retrieving data from a set of processes containing at least all those ones that have been present in the system during the whole query lifetime. If the system is prone to churn, it is easy to show that a query cannot deterministically satisfy interval validity. In this paper we propose a novel algorithm that can be used to support distributed queries by increasing the probability of a query to satisfy interval validity. The algorithm strives to (i) reduce the query calculation time (to reduce the net effect of churn) and to (ii) increase the robustness of the overlay network it builds by clustering nodes into cliques of limited size in order for their implementation to be still practical. The paper provides a set of experiments that show the tradeoffbetween the churn rate and the number of times the interval validity is satisfied.

Keywords

Distributed query answering, Interval validity, Dynamic distributed systems.

1. INTRODUCTION

During the last years, internet-based services and applications have experienced a huge growth. Nowadays, such applications (e.g. social networks or collaborative information platforms) count millions of users, connected simultaneously, that produce and consume information bringing their personal contribution to the application life. Such huge amount of data represents the real added value of these applications and their management through centralized platforms is not a feasible solution (i.e. it does not scale).

In fact, real systems are built as highly distributed platforms based onpeer-to-peer(p2p) networks orcloud technologies, where data are spread over a large number of machines connected through network links. In such context, one of the main challenges is represented by the information retrieval: obtaining correct, complete and accurate results, for a query involving the participation of thousand of processes storing a huge amount of data, is actually a very complex task. In order to provide valid query answers, data stored locally at each node have to be extensively checked and the query result is the collection of all the data matching the conditions expressed in the query. However, this procedure is expensive and, in distributed systems characterized by the continuous arrival and departure of processes (phenomenon also known aschurn), it is not efficient. In fact, processes behave autonomously by joining and leaving the system at their will without necessarily completing the ongoing executions of a distributed algorithm. As a consequence, the query result can be approximated. As an example, consider an aggregate query(e.g. sum, count, average, max and min) where the result is a single value computed by collecting the contributions of all the processes. Providing an answer to such queries in presence of churn is a challenging task. Bawa et al. introduced in [1] three validity conditions for aggregate queries to formally define which are the admissible results when they are executed in a distributed system prone to continuous churn.

The first validity condition, namelysnapshot validity allows the aggregate query to return any value such that it is the result of the query executed on some snapshot of the network taken during the query execution periodInterval validity allows the aggregate query to return a value computed by taking into account at least the contributions of all the processes part of the system for the entire query period. Finally, single site validity allows the query to return a value computed by taking into account at least all the contributions of processes connected for the whole query execution period. Note that, in a distributed system where processing times and network latencies are not negligible, the query execution takes a certain amount of time to be completed and, during this period, the system changes its composition due to the churn action. As a consequence, the authors have shown that snapshot validity and interval validity cannot be deterministically achieved if the network topology is arbitrary (i.e. the network appears as a random graph) and the system is dynamic.

In this paper, we introduce a novel overlay network structure to support aggregate queries execution satisfying interval validity with high probability, in a distributed system prone to continuos churn. The proposed overlay has a tree-based structure where vertexes of the tree are represented byvirtual nodes. A virtual node is a set of processes such that each of them is connected to each other (i.e. processes composing the virtual node are connected through a clique graph) and, given two virtual nodes, they are connected establishing multi-edges.

Once the overlay is set up, it has to be maintained due to the continual arrival and departure of nodes; thus, we provide an algorithm responsible of the overlay maintenance. In particular, the

algorithm includes a *balance* procedure to (i) maintain the size of virtual nodes almost the same and (ii) to maintaining the tree with the lowest diameter.
Finally, we provide an experimental evaluation of the proposed approach showing that, interval validity can be achieved with high probability.

The rest of this paper is organized as follows: Section 2 introduces the distributed query answering problem in distributed environment, Section 3 presents the definition of the overlay network structure and an algorithm for its maintenance, Section 4 provide an experimental evaluation. Finally, related works are discussed in Section 5 and then the conclusion.

2. ANSWERING QUERIES IN DISTRIBUTED SETTINGS

2.1 System model

The distributed system is composed, at any time, by a finite set Π of processes each of which is characterized by a unique identifier.
Processes part of the distributed system change along time, due to join of new processes and voluntary leave of processes already part of the system (i.e. the distributed system is affected by continuos *churn*). As a consequence of the churn, the size of the distributed system can change unpredictably.
All processes that are part of the distributed system, even for a very short period of time, constitute the overall system population and this population can be composed by an infinite number of processes.
Processes belonging to the distributed system can communicate by exchanging messages through symmetric reliable communication channels. Messages can experience unpredictable but limited delays due to transmission latencies.
When a process p_i wants to join the distributed system, it executes a *join procedure* that, starting from a *bootstrap process*, provides p_i with a list, called *local view*, containing the identifiers of a subset of processes part of the system.
Given the set of processes part of the system, together with their local views, we obtain an *overlay network*. This overlay network is a logical network that can be used to route application messages among the processes in Π by using the logical links represented by the identifiers contained in the local views. The number of identifiers contained in each view is limited to avoid scalability issues. An initial reference to a bootstrap process can be obtained by querying an external bootstrap service [8] that returns the identifier of a process in Π.
Processes can leave the system autonomously and independently; when a process leaves the system, it does not perform any specific procedure, it just stops receiving and sending messages; as a consequence, a voluntary leave can be considered as a failure. For ease of presentation, in the following we will use the term leave to indicate both voluntary leaves and failures. When a process leaves, it disappears from the overlay network and processes having its identifier in their local view has a dead entry.

2.2 Calculating the answer for a query

Given a query issued on a distributed system, its result can be computed by taking some basics steps. In particular, given a query q, it has to be propagated to all the processes part of the system that will evaluate it and then will return back the result to the querying node. In the case of aggregation queries, the query initiator will also aggregate all the received contributions. These steps are generally referred as in network processing for query answering [3]. Note that, while the techniques used to locally evaluate queries strictly depends from the type of query, the query dissemination in the whole system and the following gathering of results are usually executed by leveraging a spanning tree of the underlying overlay network. Such a tree can be pre-configured or built on-line when the query is submitted. Whatever is the technique used to build and maintain the dissemination tree, the query is first spread through a broadcast operation and then the results are collected in a following convergecast phase.

2.3 The validity of answers in dynamic settings

Given a distributed system Π, composed by a set of processes that does not change for a sufficient long time (i.e. the time needed from the query to be disseminated and the results to be collected), it is possible to adopt the approach described in the previous section without having any problem. In fact, all the processes receiving the query will answer with the result coming from their local evaluation and the final result of the query will be valid as no failure or leave is experienced during the query execution.

However, real systems are characterized by the presence of processes that autonomously decide whether join or leave. Thus, if Π changes during the query execution, defining the meaning of "valid" for a query result is a tricky task. Bawa et al. provided in [1] a set of *validity properties* that can be used to characterize when the result of an aggregate query, executed in a dynamic system, is valid. In particular, the authors observed that during the execution of a query the system can undergo various changes that make it pass through several configurations. A generic configuration, identified as Π_i, is defined by the set of processes belonging to the distributed system. Given two following configurations, Π_i and Π_{i+1}, they differ for one process that left or joined that system.

Given a query and the set $C = \{\Pi_i, \Pi_{i+1}, \ldots \Pi_{i+j}\}$ of all the system configurations experienced during its execution, the interval validity property can be defined as follow:

Definition 1. *A query is said to be interval valid if its result is calculated on a set of processes H such that $\cap_{x\in[i,i+j]}\Pi_x \subseteq H \subseteq \cup_{x\in[i,i+j]}\Pi_x$.*

Intuitively, interval validity is satisfied when the result is calculated considering at least the contributions coming from the set of processes part of the system for the whole query execution period; nothing is said about processes that leave/join the network while the query is already running and the final result *can* also include/miss data coming from newly joined or leaving processes.

Bawa et al. showed that interval validity cannot be achieved deterministically in a distributed system without limiting the amount of churn. Note that, Interval validity is violated only if some processes that remain in the system for the whole query execution are, at some point, isolated from the rest of the system, due to the disconnection of other processes that are in the path connecting them to the query initiator. Let us observe that the probability of disconnection is related both to the ability of the overlay network management protocol to maintain stable connections among processes in the system and to the churn rate.

From the above observation, it is possible to derive the following general principles useful to increase the probability to have interval query:

• disconnection of processes in the query spanning tree should be avoided; only leaf processes are allowed to leave the system during the query execution as their departure will not affect the computation;

• the query execution period should be reduced as much as possible; given a specific churn rate, in fact, reducing the computation

time helps in reducing the total number of processes that join or leave the system, thus lowering the overall probability that one of these actions will render the final answer non interval valid.

3. ALGORITHM DESCRIPTION

In order to improve the probability for query answers to be interval valid despite the presence of churn we should thus leverage the aforementioned principles. Note that a simple tree based structure (the one usually employed to solve such kind of tasks) could be easily tuned in order to satisfy the second principle by increasing the degree of each father node, consequently reducing the overall tree height. However such a structure is inherently fragile and can be easily destroyed in presence of even low churn rates.

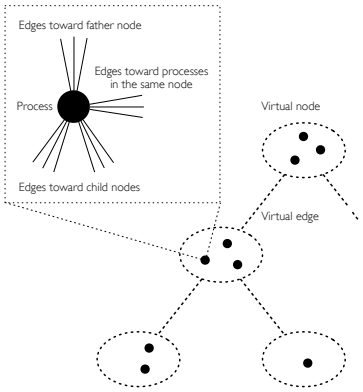

Figure 1: Architecture of the overlay

In order to overcome these limitations in this section we propose a novel overlay network whose internal structure has been purposely designed to meet the two principles. The main idea behind our solution is to organize processes in a tree-based structure (Figure 1) where each node of the tree is a *virtual node* constituted by several processes. Each virtual node is a robust structure that can be used to make the overall overlay compliant with the first principle; by smartly managing the presence of processes in various virtual nodes we can reduce the probability that non-leaf virtual nodes disappear during the query computation.

Queries can be computed on this structure by forwarding them from the root node to all the virtual nodes and within them too, such that each process is reached and can contribute to the final answer.

3.1 Architecture overview

Each virtual node in the overlay is constituted by a set of processes that are tightly interconnected to form a clique. This kind of topology makes the virtual node resistant to node departures, such that it will not be possible to assist to overlay disconnection within a same virtual node. Obviously this strong reliability comes at the cost of increased network resource usage.

Different virtual nodes are interconnected through *virtual edges*. Every virtual nodes can have up to K children nodes, with the exceptions of the leaves. Each virtual edge is a *multi-edge* constituted by several overlay network edges that connect all processes in a virtual node to processes in a second, distinct virtual node. Again, this interconnection scheme has the sole purpose of reducing to a bare minimum the probability of disconnections within the overlay due to leaves.

As a consequence, as shown in the detail of Figure 1, each process locally maintains different sets of links: (i) a set of links toward sibling processes in the same virtual node, (ii) a set of links toward processes in the father virtual node and (iii) up to K sets of links toward processes belonging to child virtual nodes. All these links represent the node's *local view* of the overlay.

Virtual nodes are labelled with a *name* and an *ID*. A virtual node name represents its position within the tree structure and can be constructed by concatenating the name of the father node (if it exists) with a string that characterizes its position in the set of children of its father node. This can be simply achieved by enumerating the children from 0 to $K-1$. A node ID is represented by a value chosen at random in a large value space such that each ID can be considered unique with high probability. Name and ID are chosen as soon as a new virtual node is created.

3.2 Details of the operations

The algorithm is fundamentally based on the maintenance of the overlay network realized by moving processes among virtual nodes and by keeping the overlay edges up-to-date. The algorithm is round-based and executed autonomously by every process in the system. It is constituted by a periodic *monitoring & balancing* task and a *join* procedure that is executed by processes willing to join the overlay.

Join. The join procedure has the aim to connect a joining process to the overlay. Connecting a process means that the algorithm should position this process in a virtual node and set up the needed edges for it to be robustly connected to the overlay. Note that virtual nodes cannot be allowed to grow indefinitely as this would make their maintenance extremely expensive resource-wise, thus hampering the overall scalability of the overlay. A system-wide configuration parameter N_{max} defines the maximum size of each virtual node.

When a new process p wants to join the overlay it inquiries a bootstrap service to obtain the network address of an access point, i.e. a process p_{ap} already part of the overlay; p sends a join request message to p_{ap} and waits. When p_{ap} receives this request it checks how many processes are currently part of the virtual node it belongs to. Three different cases are possible:

- if the size of p_{ap}'s virtual node is smaller than N_{max}, then p can be accepted; therefore, p_{ap} sends back an acceptance message containing a copy of all the edges in its local view;
- if the size of p_{ap}'s virtual node is equal or larger than N_{max}, the join request is forwarded to one of the virtual node's children with probability C/K, where $0 \leq C \leq K$ is the current number of children; p_{ap} selects one of the children at random and forwards the join request to one random process in this child node. With probability $1 - C/K$, p_{ap} sends back to p an answer instructing it to build a new child node with a given name and ID. This answer contains also a copy of all the edges from p_{ap} pointing to processes in its same virtual node.

Eventually, p will receive a positive answer from a process that instruct it either to join an existing virtual node or to create a new one; p will use the edges provided in the response to correctly populate its local view. In order to keep the local view of all nodes as much as possible up-to-date, p will inform about its presence all the nodes connected through edges in its local view.

Note that, due to concurrency of the join, two processes could possibly instruct two distinct joining nodes to create a new child with a given name but with two different IDs. As a consequence, the overlay could end-up having a partitioned virtual node. This issue is solved by processes in the father virtual node: as soon as one of them detects the inconsistency by checking the nodes different IDs it will instruct processes in child node with lower ID to kill the node by joining again the overlay.

Monitoring & balancing. Our algorithm treats processes leaving the overlay as silent faults, therefore a monitoring mechanisms is needed to maintain the healthiness of local views. This mechanism is realized through a simple heartbeat-based approach: overlay edges are kept alive through a continuous lightweight exchange of heartbeat messages between the connected nodes (only during periods of inactivity for the edge); when a message is not received before a timeout expires, the edge is simply dropped. The timeout should be adequately designed depending on the peculiar characteristics of the physical network where the system is deployed.

The disconnection of a process from the overlay reduces the size of one of its virtual nodes. However, virtual nodes that are not leaves of the tree structure should never allow their size to drop below a threshold ; if this happens some virtual node could possibly disappear leaving the tree structure partitioned and thus the overlay network disconnected. This is something that can severely hamper the system capabilities and should be avoided as much as possible.

Our algorithm reduces the probability of such disconnection to take place by allowing processes to migrate among virtual nodes. The idea is that virtual nodes whose size drops below a threshold N_{min} become *attractors* for processes belonging to their children nodes; by using a probabilistic approach some of the processes in the children node are attracted by the father virtual node and migrate toward it. When the size of a virtual node is brought back to N_{max} it stops attracting processes. Through this approach, processes tends to migrate toward the upper levels of the structure as long as there is space available in the virtual nodes. As a consequence, at runtime only leaf virtual nodes are expected to contain less than N_{min} processes for long periods of time because they are not able to attract any process due to the lack of children.

This mechanisms is also employed to maintain the tree structure balanced (as required by the second principle from Section 2.3). Each virtual node leverages the information contained in query answers to calculate the size of each of its children nodes. When the size difference between two children is larger than a threshold δ, the virtual node starts to attract processes from the larger child and redirects these processes toward the smaller child. This process continues until enough processes migrated to the smaller child and the size different drops below δ. In this way the various branches of the tree structure tend to converge at run-time toward a common depth. Having the tree structure balanced is fundamental in order to reduce the number of links that must be traversed by each query from the root till the leaves of the tree and then back toward the root. Both the size of virtual nodes, defined by the thresholds N_{max} and N_{min}, and their degree K, has a strong impact on the overall scalability of the system. The number of edges in the local view of a process is bounded by $(K + 2) \cdot N_{max}$ (this upper bound can sometimes be violated for short periods of time if a virtual nodes grows larger than N_{max}). While choosing large values for N_{max} and K can both increase the reliability of virtual nodes and reduce the overall depth of the tree, it can impose too much burden on the monitoring mechanism as local views could grow too much. Therefore, It is advisable to choose these two parameters on the basis of (i) the expected dynamics of the system (i.e. maximum churn rate) and (ii) quality of available resources both on nodes (computational power) and on the network.

4. EXPERIMENTAL EVALUATION

In this section we report the results of an experimental evaluation based on simulations that show how much our solution is able to support distributed queries by delivering interval valid answers in various scenarios with different levels of dynamics.

4.1 Setup

The proposed algorithm has been evaluated in scenarios characterized by up to 2500 processes concurrently connected in the overlay network.

In a dynamic P2P system churn (i) is always present [11, 12] and (ii) it changes the population size between an upper and a lower bound [6]. We tried to capture both aspects proposing two different scenarios:

Constant size scenario. The system is kept at a constant size while a continuous churn rate C is applied; in this case churn acts by replacing processes: each leave of a process in the system is followed by the join of a new one.

Variable size scenario. The system, starting from a population of 2500 processes, undergoes a initial phase where processes are only removed, then it passes to a second phase where processes are added until it reaches again the initial population. During the two phases the rate of addition/removal of processes is C.

With churn rate C we define the global rate at which join and leave operations occur: if n is the initial number of processes in the system, at each time unit $n \cdot C \cdot t$ processes invoke the join operation (*IN-churn*) and $n \cdot C \cdot (1 - t)$ processes invoke the leave operation (*OUT-churn*). The value $0 \leq t \leq 1$ represent how join and leave operations are balanced: for the constant size scenario we set $t = 0.5$ (balanced IN/OUT-churn), while in the variable size scenario $t = 0$ was set during the first phase (OUT-churn only) and $t = 1$ was set during the second one (IN-churn only). Process that leave the system are chosen uniformly at random fro the current population.

In the tests we decided to stress the ability of our algorithm to support large rates of churn without suffering from disconnections. Therefore, we set the values $N_{min} = 15$ and $N_{max} = 25$ that gives us fairly large and robust virtual nodes. In order to avoid stressing too much the maintenance mechanism we decided to limit the local view size by imposing $K = 2$. With such configuration each process is expected to handle up to 100 edges, a number that is reasonably manageable also by non powerful machines [2].

Tests were run on an ad-hoc round-based simulator. Each simulated process can exchange messages with all of the other processes (i.e. the simulated underlying network is fully connected); according to the system model messages are delivered within unpredictable but bounded delays. In order to simplify the simulations during the tests we fixed the latency bound such that each message is delivered at most within the next round. We expect that increasing this bound the algorithm execution would suffer more strongly from concurrency issues; however, the analysis of the corresponding impact is left for future work.

In the tests we analyze the property of interval validity measuring the percentage of interval valid queries in function of the churn level. During the simulation queries are triggered by a special process in the root virtual node each time the answer to the previous query is returned. The query we considered is very general: it consists in collecting all the identifiers of the processes in the system. The query initiator process is a special process that is always connected in order to store the whole trace of the execution. Interval validity is checked by taking a snapshot of the system at the beginning and at the end of each query executions; Combining the results of the query with the two collected snapshots it is possible to state if the answer is interval valid.

Interval validity can be violated in two cases: (i) the overlay lose a non-leaf virtual node, thus its sub-tree is not more able to contribute to the query answer, and (ii) some erroneous virtual nodes is destroyed due to concurrency issues and their processes are thus

forced to join again the system. When the latter case happens due to massive concurrent connections to the overlay, large sets of processes could be forced to migrate in the tree. The processes involved in the migration could not be reached by the aggregation procedure thus leading to interval validity violation. In the following we will refer to the former case as *connection fault* and to the latter as *concurrency fault*. Note that connection faults are strictly related to the OUT-churn while concurrency faults are strictly related to IN-churn.

4.2 Tests in the constant size scenario

In this test we analyze a population characterized by a constant size in presence of churn, i.e. IN-churn equals OUT-churn. The purpose of this test is to find the limit above which the system is no more able to provide interval valid answers with high probability. We conducted several test by varying C from 0 to 0.1. Studies based on traces from real peer-to-peer applications showed that realistic churn rates are close to 10^{-4} additions/removals per time unit, therefore we expect the rates we used to safely cover a large spectrum of realistic scenarios.

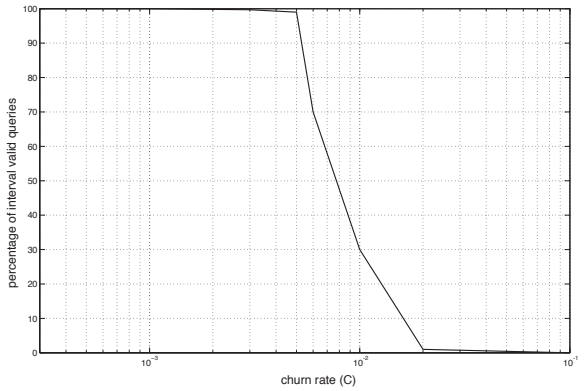

Figure 2: Percentage of interval valid queries versus churn rate in the constant size scenario.

Figure 2 reports the results of these test obtained from several runs. Confidence intervals (not showed in the graph) were always below 5%. The curve shows a peculiar bi-modal behaviour of the system. When C ranges from 0 to 0.002 the overlay efficiently manages the arrival and the departure of processes, and all the queries satisfy the interval validity property. When C ranges from 0.002 to 0.004 some limited burst of IN-churn impose migrations in the overlay that compromise the interval validity of a small number of queries; however, the percentage of valid queries is still close to 99%.

From $C = 0.004$ to $C = 0.02$ the algorithm abruptly loses its ability to fix the damages induced to the overlay by churn; the rate of joins and leaves is so large that migration does not happen quick enough to let processes participate correctly to queries. Note that while in the range from $C = 0.004$ to $C = 0.008$ the algorithm is still able to maintain the overall overlay connectivity, even if interval validity is not preserved, starting from $C = 0.008$ the overlay starts to collapse rapidly due to severe partitioning.

Finally, when C is larger than 0.02 the overlay is just a set of disconnected processes that are no more able to constitute a stable structure.

4.3 Tests in the variable size scenario

In this test we analyze a population characterized by an variable size in presence of churn. The test consists in (i) reducing the population size by 50% of its initial size by setting $t = 0$ (i.e. OUT-churn rate is $n \cdot C$) and then (ii) increasing it back to the initial size by setting $t = 1$ (i.e. IN-churn rate is $n \cdot C$). In the following we will refer to the two settings as descending and ascending phases. The purpose of this test is to analyze the algorithm behaviour during the characteristic size oscillation of many popular peer-to-peer systems by checking the ability of the system to provide interval valid queries separately during the descending and ascending phases.

The duration of both phases is linked to the churn level: the larger is the churn level and the shorter is the duration of each phase. More specifically, the duration in round R of each phase and the churn rate C are connected by the expression $C \cdot R = 2/3$. For coherence we adopted in this test the same churn levels used in the constant size scenario. In real peer-to-peer applications fluctuations in the population size are usually slower than those experimented in our tests and, moreover, they are usually limited to 20-30% of the population [6].

It is important to note that during the ascending phase only concurrency faults can take place because this phase is characterized by IN-churn only. Conversely, during the descending phase it is possible to observe both connectivity and concurrency faults; connectivity faults are the direct consequence of virtual node disconnections caused by massive amounts of leave operations. However, when processes detect their disconnection from the overlay they spontaneously try to join again the system; this non negligible amount of concurrent joins can induce concurrency faults.

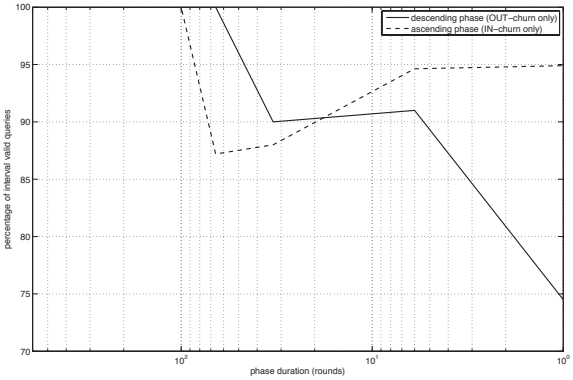

Figure 3: Percentage of interval valid queries versus churn period duration in the variable size scenario.

Figure 3 reports the results of this test. The graph show two curves related to performance observed separately during the descending (continuous line) and the ascending (dashed line) phases.

During the ascending phase the algorithm must only manage join operations that can induce concurrency faults. As long as the phase duration is large enough from 1000 to 100 rounds, two subsequent join operations are separated enough in time in order for the algorithm to avoid concurrency issues; as a consequence all the queries end with interval valid results. When the phase duration shrinks to 60 rounds concurrency faults kick-in: in this range concurrency is strong enough to cause some processes to migrate within the overlay and consequently affecting running queries. However, if the the phase duration shrinks below this point we can observe a less intuitive behaviour: the percentage of interval valid queries raises up until reaching an asymptotical value; this behaviour is justified by the fact that by reducing the phase duration we have shorter and shorter burst of intense churn; when the burst ends the algorithm employs a certain amount of time to manage the migration of some of the newly joined processes and this time depends only from the amount of them; the queries whose results can be negatively af-

fected by these migrations are only those that take place during the period that starts from the beginning of the phase and that ends when the last migrating process finally joins a virtual node (with the exception of the first one due to the definition of the interval validity property). Therefore, the shorter is the duration of the burst, the lower will be the total amount of queries that can be affected. Intuitively, the larger is the churn level, the larger is the probability to have concurrency faults; however, at the same time, the shorter is the ascending phase duration, the shorter is the time required to manage process migrations; it thus better to have large amount of joins concentrated in short burst instead of moderate IN-churn for longer time intervals.

Let focus now on the descending phase. When the phase duration is larger than 100 rounds churn is quite limited and the algorithm is able to organize the overlay preserving its connectivity. When the phase duration shrinks below this point, churn starts to heavily affect the overlay and the algorithm is no more able to prevent connectivity faults. Connectivity faults can lead to concurrency faults due to disconnected processes that try to join again the overlay; this effect is shown by the discontinuity in the curve when the phase duration is between 6 and 30 rounds: here we find again the strange behaviour previously observed in the ascending phase. However, when the descending phase duration drops below 6 rounds connectivity faults become so intense that in the end the overlay network becomes completely fragmented and the percentage of interval valid queries drops rapidly.

5. RELATED WORK

To the best of our knowledge the first work proposing an approach based on the usage of virtual nodes in an overlay network [7] where the *Kelips* algorithm is introduced; in that work Gupta et al. suggest an hybrid overlay approach where groups of processes are interconnected by a structured overlay network; these groups, formed by possibly large amounts of processes, are maintained by means of a gossip-based algorithm. Conversely, our algorithms defines groups (virtual nodes) that are maintained as cliques of few (up to few tens) processes.

More recently the authors of [9] suggested the usage of virtual nodes in order to enhance the resiliency of a DHT to the churn phenomena; In Overnesia the way virtual nodes are employed is quite different with respect to our approach: the nodes are constituted by few processes and each processes cannot move from the virtual node it has been assigned to. Differently from this approach we leverage attraction and node migration to keep the system balanced and ti promptly repair damages caused by churn.

From a general point of view small clusters of tightly connected processes have been employed to solve widely different problems. Keidar et al. addressed in [4] employed this approach in the area of sensors network: the aggregated result of a sensors network could be biased by few byzantine sensors, but having a cluster of processes it is possible to filter out malicious data before computing aggregates. The authors of [5] and in [10] propose mechanisms to distribute database functionalities over a peer-to-peer network; in particular, in [5] Furfaro et al. try to answer historical queries by leveraging gossip techniques; the authors suggest to use small cliques responsible specific data pieces in order to preserve the persistency of the data in the network.

6. CONCLUSION

Calculating answers to distributed queries in a dynamic setting where processes can join or leave the system at their will is a challenging task as shown in [1]. In this paper we introduced a novel algorithm that supports the execution of distributed queries by maintaining an overlay network. This overlay is managed and structured with the main objective of increasing the probability of query results which are interval validity. This is the first overlay network at the best of our knowledge instrumented to this specific purpose. As reported in our experimental study, interval validity can be be safely assumed to be satisfied with high probability as long as the churn rate that affect the system is below a threshold. When this threshold is surpassed, system performance rapidly degrade to a level where query answers cannot be considered complete. Let us finally remark that when the churn rate and the churn shape have a similar behavior than real churn traces (sequence of descending and ascending phases), our overlay exhibits good performance in terms of queries returning interval valid results.

7. REFERENCES

[1] Mayank Bawa, Aristides Gionis, Hector Garcia-Molina, and Rajeev Motwani. The price of validity in dynamic networks. *Journal of Computer and System Sciences*, 73(3):245–264, 2007.

[2] Kenneth Birman. *Building secure and reliable network applications*. Manning Publications Co., 1997.

[3] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards sensor database systems. In Kian-Lee Tan, Michael J. Franklin, and John C. S. Lui, editors, *Mobile Data Management*, volume 1987 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2001.

[4] Ittay Eyal, Idit Keidar, and Raphael Rom. Distributed clustering for robust aggregation in large networks. In *HotDep09*. IEEE, 2009.

[5] Filippo Furfaro, Giuseppe M. Mazzeo, and Andrea Pugliese. Managing multi-dimensional historical aggregate data in unstructured p2p systems. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints), 2009.

[6] Saikat Guha, Neil Daswani, and Ravi Jain. An experimental study of the skype peer-to-peer voip system. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, 2006.

[7] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[8] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. The bootstrapping service. In *Proceedings of the International Workshop on Dynamic Distributed System (IWDDS)- ICDCS Workshops*, page 11, 2006.

[9] João Leitão and Luís Rodrigues. Overnesia: a resilient overlay network for virtual super-peers. Technical Report 56, INESC-ID, December 2008.

[10] Norvald H. Ryeng and Kjetil Nørvåg. Robust aggregation in peer-to-peer database systems. In *IDEAS '08: Proceedings of the 2008 international symposium on Database engineering & applications*, pages 29–37, New York, NY, USA, 2008. ACM.

[11] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking (MMCN)*, January 2002.

[12] S.Rhea, D.Geels, T.Roscoe, and J.Kubiatowicz. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference (ATEC '04)*, pages 10–10, 2004.