Distrib Parallel Databases (2012) 30:1–26
DOI 10.1007/s10619-011-7086-7

Satisfaction-based query replication

An automatic and self-adaptable approach for replicating queries in the presence of autonomous participants

Jorge-Arnulfo Quiané-Ruiz · Philippe Lamarre · Patrick Valduriez

Published online: 1 October 2011

Abstract In large-scale Internet-based distributed systems, participants (consumers and providers) are typically autonomous, i.e. they may have special interests towards queries and other participants. In this context, a way to avoid a participant to voluntarily leave the system is satisfying its interests when allocating queries. However, participants satisfaction may also be negatively affected by the failures of other participants. Query replication is a solution to deal with providers failures, but, it is challenging because of autonomy: it cannot only quickly overload the system, but also it can dissatisfy participants with uninteresting queries. Thus, a natural question arises: *should queries be replicated*? If so, *which ones*? and *how many times*? In this paper, we answer these questions by revisiting query replication from a satisfaction and probabilistic point of view. We propose a new algorithm, called S_bQR, that decides on-the-fly whether a query should be replicated and at which rate. As replicating a large number of queries might overload the system, we propose a variant of our algorithm, called S_bQR+. The idea is to voluntarily fail to allocate as many replicas as required by consumers for low critical queries so as to keep resources for high critical queries during query-intensive periods. Our experimental results demonstrate that our algorithms significantly outperform the baseline algorithms from both the

Communicated by M. Tamer Özsu.

J.-A. Quiané-Ruiz (✉)
Saarland University, 66123, Saarbruecken, Germany
e-mail: Jorge.Quiane@cs.uni-sb.de

P. Lamarre
LINA—University of Nantes, 44322, Nantes, France
e-mail: Philippe.Lamarre@univ-nantes.fr

P. Valduriez
INRIA and LIRMM, 34392, Montpellier, France
e-mail: Patrick.Valduriez@inria.fr

performance and satisfaction points of view. We also show that our algorithms automatically adapt to the criticality of queries and different rates of participant failures.

Keywords Fault-tolerance · Query replication · Participants satisfaction · Autonomous participants · Self-adaptation · Query criticality

1 Introduction

In the last decade there has been a considerable increase in computing resources requirements in different research fields as well as in the industry. These needs have motivated the development of new distributed systems allowing users to share data, services, or computing resources using Internet as a large virtual computer. Volunteer computing (such as BOINC and distributed.net) is only one example of these large-scale distributed systems. A particularity of such large-scale systems is that participants—consumers and providers—are autonomous in the sense that they may leave and join the system at will, but also, that they may have special interests (*intentions*) for some queries[1] and other participants. For example in BOINC, a system that allows scientific communities to create applications by using computing resources of thousands of volunteers across the world, a consumer may desire to receive results from highly reputed providers while a provider may desire to perform queries for some preferred projects.

In this context, satisfying participants, i.e., to fill their intentions, is quite important when allocating queries. This is because dissatisfaction might lead participants to leave the system, which in turn causes some loss of system functionality and capacity to perform queries. But also, the departure of a participant may yield other participants to leave the system in a domino effect [18]. As a result, many solutions have been proposed to deal with different query processing problems in the presence of autonomous participants, e.g., [12, 14, 18, 24]. However, participants availability is usually not addressed, which is crucial because provider failures can significantly dissatisfy consumers with no results for their queries and consumer failures can dissatisfy providers as their results cannot be returned to failed consumers. Therefore, as participant failures are the rule in large-scale distributed systems [4], the responsiveness of applications built on top of autonomous participants is increasingly limited by the availability of participants rather than performance.

A basic solution to deal with provider failures is to re-allocate, after detection of a provider failure, the query to another provider. This approach, however, can significantly increase response times. An alternative solution is then query replication [1–3, 9, 10, 25]. Notice that query replication can be done by users or the system. Each of them with different goals in mind. On the one hand, users usually replicate queries in order to deal with byzantine providers. On the other hand, the system can decide to replicate queries in order to guarantee the number of results asked by users under participant failures.

[1] We use the word "query" in the general sense of service request in information systems, thus with a more general meaning than query in databases.

In this paper, we focus on query replication for dealing with participant failures (i.e. done by the system), which can be *passive* or *active*. Passive query replication (which is based on checkpointing or logging techniques [7, 8]) is not appropriate for dealing with autonomous participants. This is because passive replication inherently assumes (because of checkpointing techniques) that providers store exactly the same information and process such information in the same way to produce results. Furthermore, passive replication can significantly increase response times, because there is a system overhead for detecting provider failures, determining which queries have been stopped, and rescheduling stopped queries. On the other hand, active query replication [9] is more adequate in this context as it does not make this assumption. It allocates queries to the number of providers required by a consumer (called primary providers) plus some other providers (called backup providers). In this way, the results produced by backup providers can be returned to consumers in case of failure of primary providers. Nevertheless, most of them support a fixed number of provider failures and none considers either consumer failures nor participants satisfaction.

1.1 Motivating example

Applications of different domains need to deal, in an automated way, with participant failures in order to operate correctly. *Volunteer computing*, *Web services*, and *grid computing* are some examples of these systems. However, a discussion on how one can apply our proposal to each of them would be too long and beyond the scope of this paper. For this reason, we illustrate query replication in the presence of autonomous participants only with an application from volunteer computing. We use BOINC as example, a system that allows scientific communities to create applications by using computing resources of thousands of volunteers across the world. The query processing principle is as follows. Applications (the consumers) submit their queries to BOINC by providing the number of providers from which they want results. Volunteers (the providers) get queries from BOINC and return their results to BOINC, which in turn returns them to consumers.

Consider now a simple scenario where a given research project running on BOINC (a consumer in the BOINC system) sends a query, with a very high criticality, requiring results from a single provider. Suppose that, when the query arrives into the system, the relevant providers (those which can treat the query) have low query load, high intentions to perform the incoming query, and high failure probability. In this case, replicating the query seems a good idea. In contrast, consider now a scenario where a consumer sends a query with low criticality requiring results from several providers. Suppose that, when the query comes in, the relevant providers have high query load, low failure probability, and low intentions to perform the incoming query. In this case, it is better to not replicate the query so as to avoid overloading and dissatisfying providers. Furthermore, by not replicating this query one can prioritize queries with high criticality.

However, in most of the cases it is not that simple and one needs to answer the following simple question: *should queries be replicated?* If so, *which queries should be replicated? how many query replicas should be created?* and *which information must be used to decide on query replication?* This paper answers these questions by

revisiting active query replication from a satisfaction and probabilistic point of view. To the best of our knowledge, this is the first work that deals with participants failures by considering the satisfaction of participants as well as the failure probability of participants at the same time.

1.2 Research challenge

For low query loads, it is easy to answer these questions since there are enough computing resources to perform queries, including backup queries. In these cases, we can thus replicate queries without fearing the consequences (except if such replication hurts participants satisfaction). However, the difficulty increases as the query load gets higher, because the load due to backup queries may induce even more significant problems than initial queries. This is because overloading the system impacts the system performance with longer response times, which in turn increases the probability that a participant fails before performing a query. Additionally, one must consider participants satisfaction as dissatisfaction might cause a massive number of voluntarily departure of participants [18]. Therefore, supporting query replication in the presence of autonomous participants is challenging for several reasons:

- The overhead of query replication may outweigh its benefits, by over-utilizing the computing resources of the system or requiring either more powerful providers or additional providers.
- A provider might not have the same satisfaction of being utilized as primary or backup provider. This is because the query results produced by a backup provider are returned to the consumer only in case the primary providers fails. Furthermore, we assume they are aware of the destination of their results. One could think that one can avoid to give this information to providers. However, this is not only unfair but also not always possible, e.g. where providers get paid for their produced results.
- On the other hand, providers also consume their computing resources for nothing in case of consumer failures.

1.3 Contributions

In summary, the proposals of this paper are as follows.

- We formalize the query allocation problem and make precise query replication in the presence of autonomous participants (Sect. 2).
- We extend the satisfaction model we presented in [18] so as to consider the criticality of queries and the potential participants failures (Sect. 3).
- We introduce a global satisfaction notion to characterize the fact that (i) queries have different criticality for consumers; (ii) a consumer may receive less results than it expects; and (iii) a provider may perform queries for nothing (Sect. 4).
- We propose two automatic query replication algorithms, S_bQR and S_bQR+, that consider global satisfaction as the basis of their functionality to decide on-the-fly (i) which queries should be replicated and (ii) how many query replicas should be created (Sect. 5).

– We experimentally demonstrate that our algorithms: (i) significantly outperform popular baseline algorithms and (ii) automatically adapt to the workload and the criticality of queries (Sect. 6).

2 Problem definition

2.1 System model

We adopt the usual architecture of a mediator m, and of a set I of autonomous participants.[2] The role of mediator m is to do all the necessary computations so as to decide which providers allocate a query. Participants may play two different roles: consumer and provider. The set of consumers and providers are denoted by C ($C \subseteq I$) and P ($P \subseteq I$), respectively. Besides autonomy, we assume that participants perform queries when they are required for and assume that they can fail, but only for network failure or a software dysfunction. To formalize this aspect, we assume that each participant $i \in I$ has a probability f_i to fail per time unit. Many research works have addressed this problem of estimating this failure probability [6, 13]. Thus, in this paper, we simply assume that the mediator is able to estimate this probability. Mediator m may also fail, but this is orthogonal to the focus of this paper. Replicating mediators is a solution to deal with this, but for the sake of simplicity we assume in this paper that m never fails.

Providers in P are potentially heterogeneous in terms of capability and capacity. Heterogeneous capability means that providers usually do not provide the same functionalities and thus cannot deal with the same queries. Heterogeneous capacity means that some providers may perform more queries per time unit than others. We denote those providers that can deal with a query q (the relevant providers) as P_q, with $P_q \subseteq P$. A consumer $c \in C$ submits a query to mediator m in a format abstracted as a 4-tuple $q = \langle c, d, \gamma, n \rangle$, where: $q.c \in C$ is the identifier of the consumer that has issued the query; $q.d$ is the description of the task to be done; $q.\gamma \in [0..1]$ denotes the criticality of the query. Indeed, it may be crucial for a consumer to receive as many results as required for some queries while it may tolerate to receive no results for other queries. Notice that as in practice consumers exactly knows what they are querying for, it is easy for them to specify the criticality of their queries. The greater the value is, the more critical the query is; $q.n \in N^*$ is the number of providers from which a consumer desires to fetch results. Among others, a consumer may desire to allocate a query q to various providers for two reasons. First, to avoid Byzantine providers by comparing their results (in this case $q.n$ is in [3..5]). Second, to compare results so as to get the result that best fits its intentions, such as in flight booking systems (in this case $q.n$ is usually greater than 5). Hereafter, we simply use c, d, n, or γ when there is no ambiguity on q.

Because of their autonomy, participants are interested in performing some queries and in the way their queries are treated. This is why, given a query q, consumer $q.c$

[2] Notice that, scaling up to several mediators is orthogonal to the problem we consider in this paper. We recently addressed this scaling up problem in [17].

(respectively, each provider that is able to perform q) gives its intentions, for getting results from each provider p in set P_q (resp., for performing q), to m. In [18], we presented a strategy for participants to compute their intentions. Hence, in this paper, we assume participants provide their intentions, in the interval $[-1..1]$, to mediator m as specified in [18]. The greater the intention value is, the greater the desire of a consumer (resp., provider) to see its query be treated by a given provider (to perform a given query) is. Mediator m stores consumer's intentions in vector $\overrightarrow{CI}_q$ and providers' intention in vector $\overrightarrow{PI}_q$. For example, $\overrightarrow{CI}_q[p]$ denotes the intention of consumer $q.c$ to see its query q be treated by provider p and $\overrightarrow{PI}_q[p]$ denotes the intention of provider p to perform query q.

Finally, notice that failures of participants may dissatisfy other participants with no results for their queries or with results that are not returned to consumers. This is why providers also express the cost of performing a query q and that consumers indicate how critical their queries are. Providers' costs are in the interval $[0..+\infty[$ and stored by m in vector $\overrightarrow{PC}_q$. For example, a cost $\overrightarrow{PC}_q[p] = 100$ could mean the number of milliseconds that provider p needs to perform a query q.

2.2 Query allocation problem

Generally speaking, the goal of mediator m is to allocate each incoming query q to a set of providers so that good system performance, high participants' satisfaction, and results for queries with high criticality are ensured. We can divide this general problem into the two following independent subproblems.

RANKING RELEVANT PROVIDERS. To allocate a given incoming query q, the mediator first ranks providers in P_q according to a policy based on the challenges the system wants to solve. For example, the mediator may score providers by considering: providers' utilization for applications requiring query load balancing [19, 20]; participants' intentions for volunteer computing applications [18]. Thus, it might exist as many scoring functions as types of applications. We assume that the mediator can provide a vector $\overrightarrow{R}_q$ of ranked providers so that $\overrightarrow{R}_q[1]$ is the best scored provider and $\overrightarrow{R}_q[\|P_q\|]$ is the worst scored provider. Once more again, mediator m can rank providers by applying the techniques we proposed in [18].

SELECTING RELEVANT PROVIDERS. The problem here is that of deciding the number of providers to which to allocate a query. Formally, given an incoming query q and vector $\overrightarrow{R}_q$, the mediator must choose r best ranked providers in $\overrightarrow{R}_q$ to which to allocate q. Set $\widehat{P}_q^r$ denotes such a set of r best ranked providers, i.e., $\widehat{P}_q^r = \overrightarrow{R}_q[1..r]$.

In this paper, we focus on the problem of *selecting relevant providers*. It is worth noting that a simple solution to this problem is to allocate a query q to the number of providers required by the consumer (which leads to have $r = q.n$). However, this approach inherently assumes that either participants cannot fail or failures are treated after detection. In the presence of autonomous participants, it is possible that only a set $\widehat{\widehat{P}}_q$ (with $\widehat{\widehat{P}}_q \subset \widehat{P}_q^r$) of providers returns results for a query q, instead of set $\widehat{P}_q^r$. We formalize this problem in the presence of autonomous participants as follows.

PROBLEM STATEMENT. Given an incoming query q and ranked vector $\overrightarrow{R}_q$ of providers, each $p \in \overrightarrow{R}_q$ with a failure probability f_p, mediator m must determine r to allocate q to $\overrightarrow{R}_q[1..r]$ providers so that participants are satisfied and that consumers receive as many results as required for their critical queries.

3 A satisfaction model for faulty participants

In this section, we extend the satisfaction model we proposed in [18] so as to consider query criticality and faulty participants.

3.1 Consumer satisfaction

A consumer can evaluate by means of its satisfaction if it gets the results it expects from the mediator. There are two kinds of satisfaction: one with respect to what a consumer expects as results from providers and another one with respect to what a consumer expects from the mediator (i.e., query allocations to its preferred providers). [16] proposes a satisfaction definition that is of the first kind. The authors define consumer's satisfaction as the average of the consumer's satisfaction in each query it has issued. Satisfaction is computed by a consumer by evaluating results with respect to response times and information freshness. [18] proposes a satisfaction definition with regards to what a consumer expects from the mediator (the second kind of satisfaction) as the average of the consumer's intentions in each query it has issued. Both satisfaction definitions use the same maths and are quite important for a consumer. However, when replicating queries, the mediator is interested in what a consumer expects from query allocations. This is why we consider the latter kind of satisfaction. Generally speaking, we define the satisfaction of a consumer as in [18]. But, we also take queries' criticality and providers' failures into consideration. Intuitively, an incoming query with $\gamma = 1$ (respectively $\gamma = 0$) means that the consumer would not be satisfied at all if it did not receive all the results it requires (resp. means that the satisfaction of the consumer strongly depends on the number of results it receives). To consider this, given a query q, we introduce the *consumer's satisfaction coefficient* w.r.t. q (denoted by Δ), which we define as the importance of the number of providers that return results. The role of this coefficient is to weight the average of consumer's intentions. We formally define this satisfaction coefficient as follows.

Definition 1 (CONSUMER SATISFACTION COEFFICIENT) Let x denote the number of providers that return results for a query ($x = \|\widehat{\widehat{P_q}}\|$), the satisfaction coefficient concerning the allocation of a query q is defined as follows,

$$\Delta_q^x = \begin{cases} \dfrac{1-\gamma}{1-\gamma \cdot \frac{x}{n}} & \text{if } \gamma < 1 \\ 1 & \text{if } \gamma = 1 \wedge x = n \\ 0 & \text{if } (\gamma = 1 \wedge x \neq n) \vee x = 0 \end{cases}$$

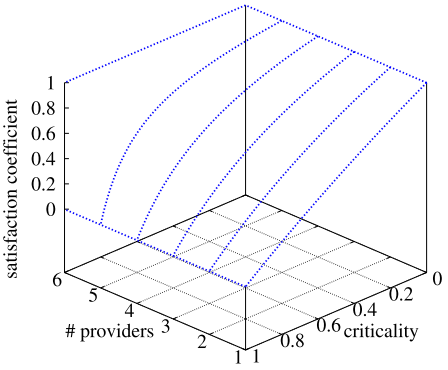


Fig. 1 Number of providers returning results vs. query criticality, when a consumer requires 6 providers

It is worth noting in (1) that when the criticality of a query takes the value of 1 and the number of providers returning results is the same as that required by the consumer, the satisfaction coefficient takes 1. In contrast, if $\gamma = 1$ and the number of results is smaller than that required by the consumer, the satisfaction coefficient always takes zero. We illustrate the behavior of the satisfaction coefficient in Fig. 1. Observe that as the query criticality increases and the number providers producing results decreases, the satisfaction coefficient decreases. This leads to a decrease of consumer's satisfaction, which is defined as the average of the consumer's intentions concerning the set of providers that return results times the satisfaction coefficient (see Definition 2)

Definition 2 (CONSUMER SATISFACTION FOR A SINGLE QUERY) Given an incoming query q, the satisfaction of $q.c$ concerning the allocation of q is given by,

$$\delta_s(c, \widehat{\widehat{P_q}}) = \Delta_q^{\|\widehat{\widehat{P_q}}\|} \cdot \frac{1}{n} \cdot \sum_{p \in \widehat{\widehat{P_q}}} (\overrightarrow{CI}_q[p] + 1)/2$$

The $\delta_s(c, \widehat{\widehat{P_q}})$ values are between 0 and 1. The closer the value from 1 is, the greater the satisfaction of a consumer is.

3.2 Provider satisfaction

A provider that has not failed can evaluate, by means of its satisfaction, if the mediator allocates queries according to its intentions. Conversely to a consumer, the fact that a query has high criticality, or not, does not influence the satisfaction of a provider. In turn, the fact that a provider performs a query and its results are not returned to the consumer may hurt its satisfaction (depending on its cost). What can hurt a provider's satisfaction is: (1) to be required to treat a query it does not desire to perform; (2) to be rejected for the treatment of an interesting query. Notice that showing its unhappiness for these cases is crucial for a provider so that it can be prioritized in the near future to get what it prefers; (3) to perform a query as backup for nothing. This is because a provider is usually selfish. Thus, spending computing resources to perform queries from which it obtains no benefit does not meet its intentions at all. As a consequent, a

relevant provider may have one of three possible states after the allocation of a given query. We formalize this in the following definition.

Definition 3 (PROVIDER SATISFACTION FOR A SINGLE QUERY) Given an incoming query q, let P_q^{ok} denote the set of providers that did not fail in the time interval required to perform q and p be a provider in $P_q \cap P_q^{ok}$. The satisfaction of p concerning q is given by,

$$\delta_s(p, \widehat{P}_q^r, \widehat{\widehat{P}_q}) = \begin{cases} \big(\overrightarrow{PI}_q[p] + 1\big)/2 & \text{if } p \in \widehat{\widehat{P}_q} \\ \big(-\overrightarrow{PI}_q[p] + 1\big)/2 & \text{if } p \in (P_q \setminus \widehat{P}_q^r) \cap P_q^{ok} \\ 1/\big(2 + \overrightarrow{PC}_q[p]\big) & \text{if } p \in \big(\widehat{P}_q^r \setminus \widehat{\widehat{P}_q}\big) \cap P_q^{ok} \end{cases}$$

Each line of the above definition corresponds to one of the three possible cases discussed early. Notice that the third line translates the cost values into the interval]0..0.5], which means that a provider always has low satisfaction when working for nothing. The provider's satisfaction values are in the interval [0..1] and the greater the value, the greater the satisfaction.

4 Global satisfaction

The main goal of replicating queries is to meet consumers demand and hence satisfy consumers. This is clearly a positive aspect of query replication. However, in the presence of autonomous participants, backup providers (those running query replicas) can see their results not be returned to the consumer if no primary provider fails. This means that backup providers utilize their resources for nothing, which may significantly dissatisfy them. This is the negative aspect of replicating queries when participants are autonomous. This is why we introduce the *global satisfaction* notion, whose goal is to compare both aspects so as to determine if it is a good idea to replicate a query. One may think that global satisfaction may be achieved by each participant being satisfied on average. This, however, is not possible as participants usually compute their satisfaction after query allocations, or even after receiving results, while decisions to replicate queries are done before allocating queries. We thus consider a global satisfaction notion that takes place before query allocations and hence it depends on the possibility that a participant fails.

To define global satisfaction, we therefore must consider all the possible cases of failure, which requires some work in probabilities. Thus, in the following, we first characterize, in Sect. 4.1, the probability that a query be successfully treated. We then provide, in Sect. 4.2, a global satisfaction definition that consider the intentions of participants as well as their failure probability.

4.1 Probabilities of success

As we assume that faults are not correlated, the probability that a participant i fails in a time unit is f_i, where f_i denotes the failure probability of a participant i. Let

t_q^p denote the time required by a provider p to perform a query q. Consequently, the probability A_q^i that a participant i does not fail in a discrete time interval t_q^p is:

$$A_q^i = (1 - f_i)^{t_q^p}$$

Given this, what is important for a consumer is to know the probability that its query be successfully treated, i.e. performed by at least $q.n$ providers. We formally define this as follows:

$$SP_q^n(\widehat{P}_q^r) = \sum_{\substack{P_q^{ok} \subseteq \widehat{P}_q^r \\ \|P_q^{ok}\| \geq n}} \left(\prod_{p \in P_q^{ok}} A_q^p \prod_{p \in \widehat{P}_q^r \setminus P_q^{ok}} (1 - A_q^p) \right) \tag{1}$$

Similarly, it is important for a provider to know the probability that its results be returned to consumers. For this, a provider p has to know the probability that no more than $n - 1$ (denoted by h) better ranked providers in $\widehat{P}_q^r$ successfully treat a query. Formally,

$$S_q^h(\widehat{P}_q^r) = \sum_{\substack{P_q^{ok} \subseteq \widehat{P}_q^r \\ \|P_q^{ok}\| \leq h}} \left(\prod_{p \in P_q^{ok}} A_q^p \prod_{p \in \widehat{P}_q^r \setminus P_q^{ok}} (1 - A_q^p) \right) \tag{2}$$

In the same spirit, but from a more general point of view, we define the probability that the results produced by a provider $\overrightarrow{R}_q[a]$ and $x - 1$ other providers in $\widehat{P}_q^r$ be returned to consumer $q.c$ as,

$$S_q^a(\widehat{P}_q^r, x) = \begin{cases} \displaystyle\sum_{\substack{\widehat{\widehat{P}}_q \subseteq \widehat{P}_q^r \\ \|\widehat{\widehat{P}}_q\| = x \\ \overrightarrow{R}_q[a] \in \widehat{\widehat{P}}_q}} \left(\prod_{p \in \widehat{\widehat{P}}_q} A_q^p \prod_{p \in \widehat{P}_q^r \setminus \widehat{\widehat{P}}_q} (1 - A_q^p) \right) & \text{if } x < q.n \\ \displaystyle\sum_{\substack{\widehat{\widehat{P}}_q \subseteq \widehat{P}_q^r \\ \|\widehat{\widehat{P}}_q\| = x \\ \overrightarrow{R}_q[a] \in \widehat{\widehat{P}}_q}} \left(\prod_{p \in \widehat{\widehat{P}}_q} A_q^p \prod_{\substack{p = \overrightarrow{R}_q[j] \\ j \leq \max(k) \\ \overrightarrow{R}_q[k] \in \widehat{\widehat{P}}_q \\ p \notin \widehat{\widehat{P}}_q}} (1 - A_q^p) \right) & \text{else} \end{cases} \tag{3}$$

4.2 Global satisfaction definition

We can now define global satisfaction with respect to the way in which the mediator allocates queries. Informally, we define global satisfaction as follows. Given a query q, global satisfaction denotes the most possible satisfaction that consumer $q.c$ and providers in P_q may have if q is allocated to a given set $\widehat{P}_q^r$. Intuitively, global satisfaction denotes the sum of satisfaction of the relevant providers plus the satisfaction of the consumer. Unfortunately, possible failures of both the consumer and

providers make this more complicated. For a relevant provider that does not fail, we must consider the following three cases:

(1) A provider is allocated a query and its results are returned to the consumer. For this, it is necessary that the consumer does not fail and that no more than h (i.e., $n-1$) better ranked providers be alive. In this case, the satisfaction of a provider is based on its intention, if the consumer does not fail, or the cost of treating the query, otherwise.
(2) A provider is allocated a query replica and its results are not returned to the consumer. This happens if at least n better ranked providers do not fail. In this case, the satisfaction of a provider is based on the cost of treating the query.
(3) A provider is not allocated a query. Here, the satisfaction of a provider is based on its negative intention. For the consumer, we simply need to consider the probability that each relevant provider has to successfully return results.

Putting everything together, we formally define the global satisfaction in Definition 4, where Δ_q^j is the satisfaction coefficient that allows us to take into account the criticality of a query q based on the number j of providers returning results (Definition 1 of Sect. 3.1).

Definition 4 (GLOBAL SATISFACTION) Given an incoming query q, the global satisfaction $\Theta(\widehat{P}_q^r)$ of allocating q to a set $\widehat{P}_q^r$ is defined as follows,

$$\begin{aligned}\Theta(\widehat{P}_q^r) = \sum_{j=1}^{r} \Big(& A_q^{\overrightarrow{R}_q[j]} \cdot (A_q^c \cdot S_q^{n-1}(\widehat{P}_q^{j-1}) \cdot \overrightarrow{PI}_q[\overrightarrow{R}_q[j]] \\ & + (1 - A_q^c) \cdot S_q^{n-1}(\widehat{P}_q^{j-1}) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[j]] \\ & + (1 - S_q^{n-1}(\widehat{P}_q^{j-1})) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[j]]) \Big) \\ & + \sum_{j=r+1}^{\|P_q\|} A_q^{\overrightarrow{R}_q[j]} \cdot -\overrightarrow{PI}_q[\overrightarrow{R}_q[j]] \\ & + A_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \sum_{a=1}^{r} \left(S_q^q(\widehat{P}_q^r, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[a]] \right) \right)\end{aligned}$$

It is clear that the term "global satisfaction" can have a much broader interpretation and may be linked to many other points, e.g. the quality of results. Exploring all the different facets that the satisfaction term might have is well beyond the scope of this paper. However, anticipating Sect. 5, the algorithms we propose are quite general and can be used with any satisfaction definition.

5 Automatic query replication algorithms

Given the global satisfaction definition we presented in previous section, one can imagine that the main goal of the system is to maximize the global satisfaction, i.e.,

to allocate an incoming query to those providers that increase the global satisfaction. However, looking for global satisfaction may have surprising bad results because this approach prioritizes the most adequate participants to the system. Looking for improving individual satisfaction so as to prioritize less satisfied participants is then a better way to proceed. Furthermore, as participants are usually selfish, the mediator also tries to improve system performance when ranking providers.

Therefore, we propose two new algorithms that respect the strategy of the mediator to allocate queries. The first algorithm, called S_bQR (standing for *Satisfaction-based Query Replication*), implements a typical query replication strategy: it aims at creating some query instances in addition to those asked by consumers so that consumers receive results from their required number of providers. The second algorithm, called S_bQR+, implements a more elaborated strategy: it considers the query instances required by consumers and decides if more or less instances must be created. Intuitively, for high workloads, S_bQR+ prioritizes highly critical queries by creating less query instances (for low critical queries) than required by consumers. A key feature of both two algorithms is their simplicity, which allows for an easy implementation in any new or existing distributed information system. We discuss in detail both algorithms in the following two sections.

5.1 S_bQR algorithm

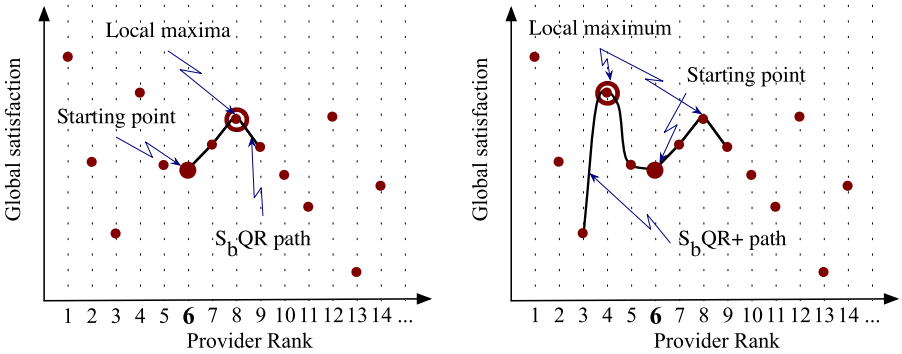
Satisfaction-based Query Replication (S_bQR for short) aims at increasing—as far as this does not decrease the global satisfaction—the probability that consumers receive results from the number of providers they require. For this, S_bQR replicates incoming queries by considering global satisfaction, that is, it only replicates a query when this yields an increase in global satisfaction. A salient feature of S_bQR is that it decides on-line which queries should be replicated and at which rate, based on both participants satisfaction and failure probability. Algorithm 1 shows how S_bQR works for a given query. The idea is simple: it creates as many backup providers as long as global satisfaction increases. In more details, given a query q, S_bQR compares the global satisfaction of the first $q.n$ relevant providers (i.e. the $\overrightarrow{R}_q[1..n]$ providers) with the global satisfaction of the $\overrightarrow{R}_q[1..n+1]$ providers. If the global satisfaction of $\overrightarrow{R}_q[1..n+1]$ is greater, S_bQR compares then such global satisfaction with that of $\overrightarrow{R}_q[1..n+2]$ and repeats the operation until it finds a set $\overrightarrow{R}_q[1..n+i]$ whose global satisfaction is higher than $\overrightarrow{R}_q[1..n+i+1]$. However, in special cases—for example, when a consumer prefers all providers at equal and providers have positive intentions towards queries—, S_bQR could finish by allocating the query to all providers, which might impact the performance of applications. To avoid this, S_bQR thus stops this process as soon as the probability that at least $q.n$ providers successfully treat the query (Equation (1)) is higher than a given threshold T. Instead of having a fixed threshold value for all queries, we take into account the criticality of queries so that the value of T be higher for critical queries. We formally define T in equation below—where max and min are the minimum values that T can take: these values are in the interval [0..1[.

$$T = \gamma \cdot (\max - \min) + \min$$

Algorithm 1: S_bQR

Input : q, $\overrightarrow{R}_q$, $\overrightarrow{CI}_q$, $\overrightarrow{PI}_q$, $\overrightarrow{PC}_q$, T
Output: $\widehat{P}_q^r$

```
1 begin
2     r = q.n;
3     while ( r < |R_q| ∧ Θ(P̂_q^r) < Θ(P̂_q^{r+1}) ∧ SP_q^n(P̂_q^r) < T ) do
4         r++;
5     return P̂_q^r;
6 end
```



(a) S_bQR case. (b) S_bQR+ case.

Fig. 2 Finding local maximum around $q.n$ (with $q.n = 6$)

In other words, S_bQR looks for the local maximum of global satisfaction as far as threshold T is not reached. Notice that S_bQR allows the mediator to satisfy participants in the long run by respecting the ranking provided by the mediator.

Example 1 We illustrate in Fig. 2(a) the principle of S_bQR when looking for such local maximum w.r.t. a query q where consumer $q.c$ requires results from 6 providers ($q.n = 6$). Assume that q has a medium criticality ($\gamma = 0.5$) and that the $6th$ best ranked provider has a higher probability of failure. Suppose now that the $7th$, $8th$, and $9th$ ranked providers have a low positive intention to perform q. Also, suppose that c has a medium and high positive intention to receive results from the $7th$ and $8th$, respectively, ranked providers, but that it has a negative intention towards the $9th$ ranked provider. In this case, S_bQR decides to create 2 backup queries because provider $\overrightarrow{R}_q[8]$ denotes the local maximum.

5.2 S_bQR+ Algorithm

We now consider the query replication from a more general point of view. That is, we not only analyze if backup queries must be created, as in the previous section, but also if the number of query instances asked by consumers—parameter $q.n$ for a query q—can be reduced so as to increase the global satisfaction. This is quite

Algorithm 2: S_bQR+

```
Input  : q, R⃗_q, CI⃗_q, PI⃗_q, PC⃗_q, T
Output: P̂_q^r
1 begin
2     P̂_q^+ = S_bQR(q, R⃗_q, CI⃗_q, PI⃗_q, PC⃗_q, T)
3     r = q.n;
4     while (r > 1 ∧ Θ(P̂_q^r) < Θ(P̂_q^{r−1})) do
5         r − −;
6     P̂_q^− = P̂_q^r;
7     if (Θ(P̂_q^−) < Θ(P̂_q^+)) then return P̂_q^+;
8     else return P̂_q^− ;
9 end
```

useful for heavy workloads when replicating queries may significantly impact the performance of applications. Then, the idea is to allocate low critical queries to less providers than those required by consumers in order to keep computing resources for highly critical queries. To do so, we propose S_bQR+, an algorithm that looks for the local maximum of global satisfaction by also analyzing global satisfaction when reducing the number of instances of queries.

We show the S_bQR+ process in Algorithm 2. S_bQR+ has the following two properties: (1) it never creates more backup queries than S_bQR, and (2) it exactly operates as S_bQR when $q.n = 1$. One can say that the instances of a query should not be reduced when its criticality is 1 since, according to the satisfaction definition of a consumer (see Sect. 3.1), the satisfaction of a consumer (regarding the job made by the mediator) falls to zero if its query is allocated to less than the desired number of providers. However, as S_bQR, S_bQR+ also works for providers and thus, in some cases, providers may benefit from the reduction of instances of a query. Besides, a consumer may be satisfied with the received results, even if it is not the desired number. In any case S_bQR+ always allocates at least one query instance so that queries be treated (i.e. $q.n \geq 1$).

Example 2 To exemplify the principle of S_bQR+ when looking for a local maximum w.r.t. a query q with $q.n = 6$, we consider again the example of previous section. But, this time we assume that q has a low criticality ($\gamma = 0.1$). As, besides creating backup queries, S_bQR+ also strives to reduce query instances if necessary, we consider those providers with a better rank than 6 (see Fig. 2(b)). For these better ranked providers, suppose that the $5th$ and $3th$ ranked providers have both a negative intention to perform q because of overload and that the $4th$ has a high positive intention to perform q. On the other side, suppose that c has a high positive intention towards all three providers. In this case, even if $q.n = 6$, S_bQR+ allocates q to the $\overrightarrow{R}_q[1..n-2]$ providers because $\overrightarrow{R}_q[4]$ represents the highest local maximum. This allows S_bQR+ to devote more computing resources to highly critical queries.

5.3 Global satisfaction computation

Both two algorithms S_bQR and S_bQR+ compare the global satisfaction of two sets of relevant providers so as to allocate the query to the set having the highest global satisfaction. To know which set of providers has the highest global satisfaction, one should compute the global satisfaction of both sets. At first glance, this comparison is complicated and may be long to realize. However, sets of providers are built according to the query allocation strategy (i.e., using vector $\overrightarrow{R}$) and hence the difference among two compared sets is always only one provider. Thus, we can reduce the global satisfaction comparison to the study of the impact of adding a provider from a given set of providers, which results in a significant simplification of the global satisfaction comparisons. We formalize this comparison in Theorem 1.

Theorem 1 *Comparing the global satisfaction of a set of relevant providers $\widehat{P}_q^r$ with the global satisfaction of the same set of relevant providers plus a new relevant provider, $\widehat{P}_q^{r+1}$, can be done by studying the impact of adding the new relevant provider only. Formally,*

$$\begin{aligned}\Theta(\widehat{P}_q^{r+1}) - \Theta(\widehat{P}_q^r) = A_q^{\overrightarrow{R}_q[r+1]} \cdot \Big(& A_q^c \cdot S_q^{n-1}(\widehat{P}_q^r) \cdot \overrightarrow{PI}_q[\overrightarrow{R}_q[r+1]] \\ & + (1 - A_q^c) \cdot S_q^{n-1}(\widehat{P}_q^r) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[r+1]] \\ & + \big(1 - S_q^{n-1}(\widehat{P}_q^r)\big) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[r+1]] \\ & + \overrightarrow{PI}_q[\overrightarrow{R}_q[r+1]]\Big) \\ & + A_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \left(\sum_{a=1}^{r} \big(S_q^a(\widehat{P}_q^{r+1}, j) - S_q^a(\widehat{P}_q^r, j)\big) \right.\right. \\ & \left.\left. \times \overrightarrow{CI}_q[a] + S_q^{r+1}(\widehat{P}_q^{r+1}, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[r+1]] \right) \right)\end{aligned}$$

Proof (Theorem 1) Our demonstration is derived from algebraic reductions of Definition 4 (the $\Theta(\widehat{P}_q^{r+1}) - \Theta(\widehat{P}_q^r)$ case). For clarity, we proceed to demonstrate equation of Theorem 1 line by line. First, in case that a provider is considered to get a query q and is also considered to be in set $\widehat{\widehat{P}}_q$, we have:

$$\begin{aligned}&\textstyle\sum_{j=1}^{r+1} \left(A_q^{\overrightarrow{R}_q[j]} \cdot A_q^c \cdot S_q^{n-1}(\widehat{P}_q^{j-1}) \cdot \overrightarrow{PI}_q[\overrightarrow{R}_q[j]]\right) \\ &\quad - \textstyle\sum_{j=1}^{r} \left(A_q^{\overrightarrow{R}_q[j]} \cdot A_q^c \cdot S_q^{n-1}(\widehat{P}_q^{j-1}) \cdot \overrightarrow{PI}_q[\overrightarrow{R}_q[j]]\right)\end{aligned}$$

Thus, all values from 1 up to r are eliminated by the subtraction because of (2). Consequently, we only consider the $r + 1$ value, that is:

$$A_q^{\overrightarrow{R}_q[r+1]} \cdot A_q^c \cdot S_q^{n-1}(\widehat{P}_q^r) \cdot \overrightarrow{PI}_q[\overrightarrow{R}_q[r+1]]$$

which demonstrates the first line (of Theorem 1). When a provider is expected to get query q and not expected to be in set $\widehat{\widehat{P_q}}$, we have the case in which consumer $q.c$ is expected to fail:

$$\sum_{j=1}^{r+1}\left(A_q^{\overrightarrow{R}_q[j]}\cdot\left(1-A_q^c\right)\cdot S_q^{n-1}(\widehat{P}_q^{j-1})\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[j]]\right)$$
$$-\sum_{j=1}^{r}\left(A_q^{\overrightarrow{R}_q[j]}\cdot\left(1-A_q^c\right)\cdot S_q^{n-1}(\widehat{P}_q^{j-1})\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[j]]\right)$$

and also the case where at least $q.n$ other providers with a ranking smaller than j are in $\widehat{P}_q^r \cap P_q^{ok}$,

$$\sum_{j=1}^{r+1}\left(A_q^{\overrightarrow{R}_q[j]}\cdot\left(1-S_q^{n-1}(\widehat{P}_q^{j-1})\right)\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[j]]\right)$$
$$-\sum_{j=1}^{r}\left(A_q^{\overrightarrow{R}_q[j]}\cdot\left(1-S_q^{n-1}(\widehat{P}_q^{j-1})\right)\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[j]]\right)$$

In both cases, values from 1 up to r are eliminated by the subtraction and hence we only consider the $r+1$ value. Consequently, we have the following equation for the first case:

$$A_q^{\overrightarrow{R}_q[r+1]}\cdot\left(1-A_q^c\right)\cdot S_q^{n-1}(\widehat{P}_q^r)\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[r+1]]$$

and

$$A_q^{\overrightarrow{R}_q[r+1]}\cdot\left(1-S_q^{n-1}(\widehat{P}_q^r)\right)\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[r+1]]$$

for the second case. The above two equations demonstrate lines 2 and 3, respectively. Now, for those providers that are expected to not get query q we have:

$$\sum_{j=r+2}^{\|P_q\|}A_q^{\overrightarrow{R}_q[j]}\cdot-\overrightarrow{PI}_q[\overrightarrow{R}_q[j]]-\sum_{j=r+1}^{\|P_q\|}A_q^{\overrightarrow{R}_q[j]}\cdot-\overrightarrow{PI}_q[\overrightarrow{R}_q[j]]$$

Notice that all values from $r+2$ up to $\|P_q\|$ are again eliminated by the subtraction. But, conversely to previous equations, the $r+1$ value remains in the right side and thus we take its negative value, which implies the following equation:

$$A_q^{\overrightarrow{R}_q[r+1]}\cdot\overrightarrow{PI}_q[\overrightarrow{R}_q[r+1]]$$

This equation demonstrates the fourth line. Finally, to demonstrate the final line (i.e. the expected satisfaction concerning the consumer), we focus on the consumer's side of Theorem 1 and thus we have the following subtraction:

$$A_q^c\cdot\sum_{j=0}^{n}\left(\Delta_q^j\cdot\frac{1}{n}\cdot\sum_{a=1}^{r+1}\left(S_q^a(\widehat{P}_q^{r+1},j)\cdot\overrightarrow{CI}_q[\overrightarrow{R}_q[a]]\right)\right)$$
$$-A_q^c\cdot\sum_{j=0}^{n}\left(\Delta_q^j\cdot\frac{1}{n}\cdot\sum_{a=1}^{r}\left(S_q^a(\widehat{P}_q^{r},j)\cdot\overrightarrow{CI}_q[\overrightarrow{R}_q[a]]\right)\right)$$

Conversely to all previous equations, even though we have repeated iterations (from 1 up to r), the subtraction cannot eliminate such values because of (3), which considers set $\widehat{\widehat{P_q}}$. In other words, $S_q^a(\widehat{P}_q^{r+1}, j)$ is different from $S_q^a(\widehat{P}_q^r, j)$ even for a same value j, which is not the case for $S_q^{n-1}(\widehat{P}_q^j)$. Therefore, we can only reduce the equation above by grouping values from 1 up to r,

$$A_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \left(\sum_{a=1}^{r} \left(S_q^a(\widehat{P}_q^{r+1}, j) - S_q^a(\widehat{P}_q^r, j) \right) \cdot \overrightarrow{CI}_q[a] \right) \right)$$

and separating the $r+1$ value,

$$A_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot S_q^{r+1}(\widehat{P}_q^{r+1}, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[r+1]] \right)$$

Thus, we have the following equation:

$$A_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \left(\sum_{a=1}^{r} \left(S_q^a(\widehat{P}_q^{r+1}, j) - S_q^a(\widehat{P}_q^r, j) \right) \cdot \overrightarrow{CI}_q[a] + S_q^{r+1}(\widehat{P}_q^{r+1}, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[r+1]] \right) \right)$$

which demonstrates the fifth line (of Theorem 1). □

5.4 Discussion

We pointed out so far that there may exist several definitions of satisfaction and that one may see the quality of results as an intuitive definition of satisfaction. However, in our context, it is not possible to include satisfaction regarding answers, This is for two reasons: (i) the evaluation of participants for a particular answer is private and, (ii) by convention, our proposal takes place before providers produce queries results (i.e. before query allocations) and hence we do not have any information about the results produced so far. Instead, we propose algorithms that are quite general and independent of the way in which satisfaction is computed. We can thus adapt the satisfaction definition to fit any particular application. This also applies to the intentions of participants, where participants may consider any information they have, such as personal experiments, reputation of participants, response time, and load. However, it is worth noting that the behavior of the system strongly depends on the way participants compute their intentions. For instance, if providers do not care about their preferences and compute their intentions by only considering their load, the system will ensure short response times.

Given the behavior of both algorithms, the reader may think that consumers can take the best providers for themselves by specifying that each of their queries is critical. Indeed, they can freely do so because of their autonomy. However, this is far from the truth. This is because this strategy does not allow consumers to differentiate what is critical for them from what is not. For low query loads, this has not a deep impact

since every query can be replicated. For high query loads, as far as they do not tell what is important for them, they simply let other participants to choose which query is replicated and which is not. Absolutely this is not what consumers expect to see. Furthermore, if all consumers set the criticality to the same value, the criticality of queries would be completely neutralized and not considered at all by our algorithms. Furthermore, dealing with this issue is responsibility of the score function itself and thus it is beyond the scope of this paper.

Moreover, the scoring function used by the mediator to produce vector $\overrightarrow{R}$ is usually based on specific demands, which are given by the application challenges that one wants to solve. As a consequence, a large number of specific query allocation methods with different behaviors may exist. For example, the score function of a query load balancing method is designed for those applications whose goal is to ensure good system performance. To deal with all this diversity, our algorithms treat the ranking function as a black box by preserving the ranking order of providers when finally deciding which providers are allocated a query. This allows us in turn to preserve the strategy chosen by the mediator to allocate queries. As a result, one can apply our approach in any kind of application.

6 Experimental study

To validate our algorithms, we compare them with the two most popular baseline algorithms. The first one is *replicateAll* [11], used for example in BOINC, which systematically generates one replica by allocating each incoming query q to $q.n + 1$ providers. The second one, used in many systems, consists to not replicate at all so that re-allocation has to deal with all problems, which we call *none*. We carry out our experimental validation with four main objectives: (i) to evaluate how well, from a satisfaction point of view, our algorithms operate in the presence of autonomous participants; (ii) to evaluate the impact on performance of backup queries generated by our algorithms; (iii) to evaluate how our algorithms operate in environments where participants have no preference; and (iv) to analyze if our algorithms can adapt to different querys' criticality and to different probabilities of participants failure.

6.1 Experimental setup

Let us first point out that the definition of a synthetic workload for environments where participants have special interests towards queries is an open problem. In [15] the authors discuss the need for benchmarks of scenario-oriented cases, which are similar to the case we consider, but this remains an open problem. Although, it is possible to consider real-world data over long periods of time from a specific application, we decided to generate a much more general workload that can be applied to different applications and environments in order to thoroughly validate our results.

We implemented S_bQA [18], which computes vector $\overrightarrow{R}$, and then implemented S_bQR, S_bQR+, and *replicateAll* algorithms on top of S_bQA. Without loss of generality, we followed [18] to define participants' intentions. That is, providers consider

their preferences, utilization, and satisfaction to compute their intentions, and consumers only consider their preferences. We do not detail this computation because it is orthogonal to the problem we address in this paper. We followed [21] to generate the experimental system.

We generated a network with 150 consumers and 300 providers who compute their satisfaction as presented in Sects. 3.1 and 3.2, respectively, and consider a single mediator. At first glance, the reader may think that this number of providers is not representative for large-scale systems. Nonetheless, we assume that all 300 providers are able to perform any incoming query, i.e. all 300 providers always compete among them in order to perform any incoming query. Therefore, this number of providers is a representative number of relevant providers found in practice.

We generated 10% of providers with *low*-capacity, 60% with *medium*, and 30% with *high*. The *high*-capacity providers are 3 times more powerful than *medium*-capacity providers and still 7 times more powerful than *low*-capacity providers. We divided the set of providers into three classes according to the interests of consumers: consumers that have *high*-interest (60% of providers), *medium*-interest (30% of providers), or *low*-interest (10% of providers).

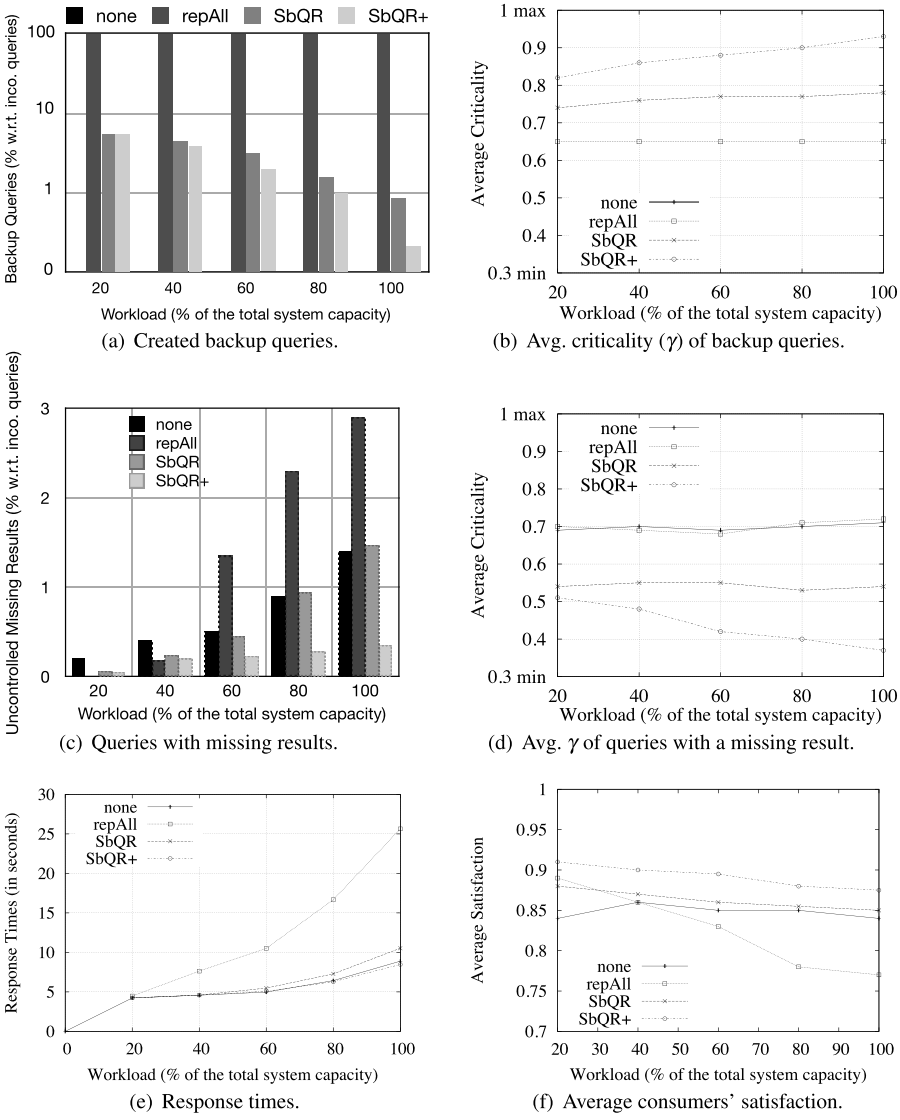
As consumers do not directly compete among them to get results from providers, it really does not matter how many they are. What impacts performance it is the workload they produce. This is why, instead of varying the number of participants, we vary the workload. We strongly believes that systems with much more participants will have the same relative performance. Notice that, the workload we generate is only with respect to the number of queries issued by consumers. Any backup query generated by the algorithms we test is added to such workload. We generated two classes of queries that *high*-capacity providers perform in 1.3 and 1.5 seconds, respectively. We assumed that queries arrive in a *Poisson* distribution. We assumed that participants have the same network capacities and that each of them has a failure probability per second of 0.03. This is a high failure probability, but we want to stress the system so as to conduct our experiments under difficult situations.

We initialized the satisfaction of participants with a value of 0.5. Then, during experiments, participants made an average of their satisfaction over the last 150 issued queries (for consumers) and the 400 queries that have passed through providers. Queries have a criticality generated at random between 0.3 and 1 and should be answered by six different providers (i.e., $q.n = 6$). Moreover, we considered that it is up to a provider p to estimate the time it needs to answer each incoming query q and gives it to the mediator. Finally, we ran 10 times each experiment and report the average of these results.

6.2 Results with autonomous participants

In these experiments, we assume that participants compute their preferences and queries cost uniformly at random in the intervals $[-1, 1]$ and $[-1, 0]$, respectively. We compute this at random for two reasons: we cannot control the environment of participants; we strive to simulate highly heterogeneous preferences of participants.

In Fig. 3(c), we observe that for low workloads *replicateAll* has (around 0.01%) less queries with missing results than the other algorithms. However, we observe in



(a) Created backup queries.
(b) Avg. criticality (γ) of backup queries.
(c) Queries with missing results.
(d) Avg. γ of queries with a missing result.
(e) Response times.
(f) Average consumers' satisfaction.

Fig. 3 Results with queries requiring six providers and for different workloads

Fig. 3(a) that it replicates 20 times more queries. The disadvantages of *replicateAll* becomes clear in Fig. 3(c) when the workload is higher than 60%: the number of queries with missing results for high workloads is twice as with our algorithms since providers are more loaded and queries are blinded-replicated. This does not happen with our algorithms, which automatically adapt the query replication rate to the workload. This is because providers take care of their load while expressing intentions. It is worth noting that S_bQR+ voluntarily aborts ∼1% of low-critical queries for high query loads in order to prioritize high-critical queries. This is why it misses a very

low percent of results for high query loads. Figure 3(b) clearly illustrates that our algorithms create much less backup queries as the workload increases. Furthermore, we observe in Fig. 3(b) that our algorithms mainly replicate critical queries. This trend is more important as the workload is higher, in particular for S_bQR+. The great advantage is that the number of missing results S_bQR+ has for non-critical queries is similar to the ones of the *none* case, while it ensures more results for highly critical queries (see Fig. 3(d)).

It is worth noting that, in the S_bQR+ case, the number of aborted queries—those which were allocated to less providers than those required—increases with the workload. S_bQR+ voluntarily aborts lowly critical queries (see Fig. 3(d)) to prioritize highly critical queries (see Fig. 3(b)). As a result, the number of queries with missing results (because of provider failure) increases much more slowly (see Fig. 3(c)). Figure 3(f) clearly shows that consumers appreciate this, where satisfaction is defined as in Sect. 3.1. Furthermore, aborting lowly critical queries allows S_bQR+ to guarantee short response times (see Fig. 3(e)). Notice that, even if S_bQR has slightly longer response times than when doing no replication (the *none* case), it significantly outperforms *replicateAll*. During our experiments, we observed that the smaller the number of required results (n), our algorithms are much better than *replicateAll*. For example, when $n = 2$, *replicateAll* starts to have problems with workloads higher than 40% while our algorithms remain stable.

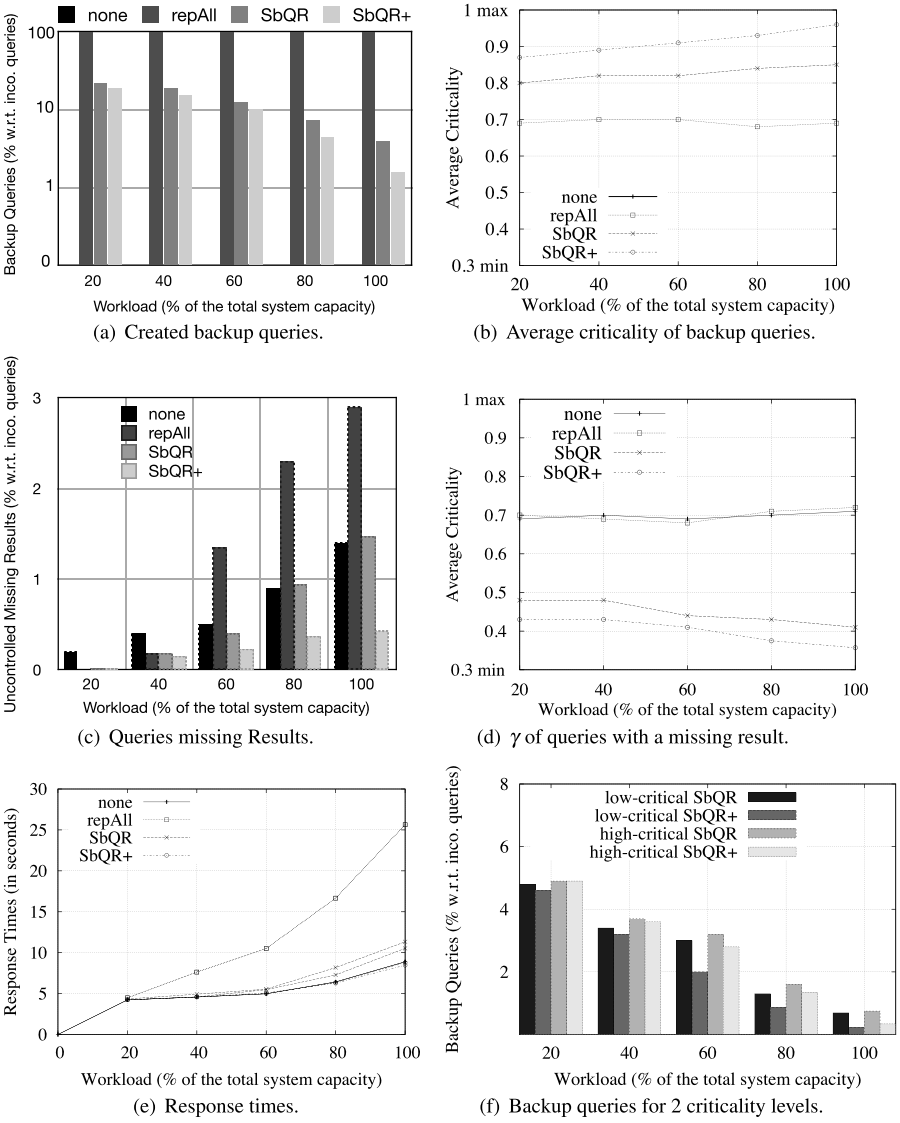
6.3 Results with passive participants

One may wonder if the previous results are impacted by the preferences of participants. To clarify this doubt, we now assume that participants are fully devoted to the system, such as nodes in a cluster. To establish this, we neutralize both preferences and costs of participants by respectively setting all of them to 1 and 0.

We illustrate the results in Figs. 4(a)–4(e). Interestingly, we observe in Fig. 4(a) that our algorithms create much more backup queries than in previous results, especially for those queries having a high criticality (see Fig. 4(b)). This is because participants have now no preference towards either queries nor other participants and hence the system only considers the criticality of queries. As a result, such an increase in the number of backup queries is reflected by having almost the same number of missing results as *replicateAll* for low workloads, and as the *none* case for high workloads (see Fig. 4(c)). Let us however highlight in these results that S_bQR+ voluntary aborts more queries than it misses due to providers failure. This proves its capacity to deal with provider failures. Now, we observe in Fig. 4(d) that our algorithms also improve their performance by preserving more critical queries. These results clearly illustrate the aim of our algorithms at mainly replicating highly critical queries. Finally, we see in Fig. 4(e) that, even though S_bQR and S_bQR+ create more backup queries, their response time is only degraded by 70 milliseconds on average. All this proves their efficiency even in systems with passive participants.

6.4 Varying criticality of queries

To analyze the sensitivity of our algorithms to different criticities of queries, we run two series of experiments: (i) with lowly critical (criticality of 0), and (ii) with highly



(a) Created backup queries.
(b) Average criticality of backup queries.
(c) Queries missing Results.
(d) γ of queries with a missing result.
(e) Response times.
(f) Backup queries for 2 criticality levels.

Fig. 4 Results with queries requiring six providers, with participants having no preferences, and for different workloads

critical queries (criticality of 1). The criticality of queries neither impacts the performance of *replicateAll* nor that of the *none* systems, because they do not consider this criteria. This is why we only show the results for our algorithms. For these experiments, we assume again autonomous participants and hence they compute their preferences and query costs as in Sect. 6.2.

Figure 4(f) illustrates the number of created backup queries for different workloads. These results confirm the fact that our algorithms tend to replicate less queries

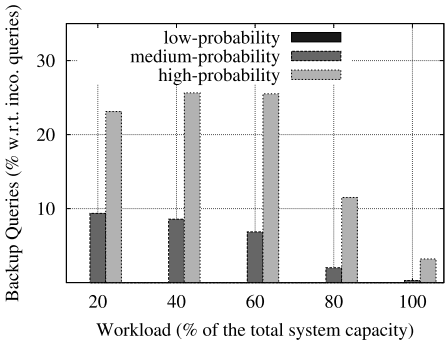


Fig. 5 Backup queries created for different workloads and two different probabilities of providers failure

as the workload increases in order to not overload providers. We can also see the sensitivity of our algorithms to the criticality of queries by creating more backup queries for the highly critical queries. We confirm that, as stated in Sect. 5.2, S_bQR+ never replicates more queries than S_bQR. In fact, as the workload increases, the number of queries replicated by S_bQR+ is much smaller than those replicated by S_bQR. Once more again, this is because providers quickly become overloaded and thus begin expressing negative intentions—which are considered by S_bQR+ to reduce the number of required providers.

6.5 Varying probability of providers failure

We finally run our algorithms in systems where providers have quite different probabilities of failure per second: 0.006 (low probability), 0.05 (medium probability), and 0.1 (high probability). We assume that queries arrive with a criticality of 1 and that consumers ask for a single answer per query, i.e., $n = 1$. We chose these values because it is with these values that providers failure impacts the more. Since $n = 1$ and because of the properties of S_bQR+ (see Sect. 5.2), the results we present here are valid for both of our algorithms. Let us say that results from these experiments are quite similar to those already observed in previous sections. For this and for space reasons, we just present the most important. In Fig. 5, we can see that, as the failure probability of providers increases, more backup queries are created by our algorithms to ensure that consumers get answers for their queries. However, when providers become overutilized, our algorithms decrease the number of backup queries. This proves the high sensitivity of our algorithms to the probability of providers failure. We could observe during our experiments that our algorithms have in average less queries with missing results than *none* and *replicateAll*. We also observed our algorithms to better satisfy participants than other algorithms. For low workloads, consumers feel better satisfied with *replicateAll*, but the difference is quite small (see Fig. 3(f)).

7 Related work

Query replication approaches can be classified in two models: *passive* or *active* query replication. In passive query replication, also known as primary-backup [5], primary

providers actively perform queries and regularly checkpoint their state to backup providers [2], which are either waiting for a checkpointing message or saving a checkpointing message. In case a primary provider fails, a backup provider takes over the role of the primary provider by reading the last checkpointed state in order to recover a state that existed before the primary provider's failure. In this way, the failure can be masked to consumers, but they can experience a long delay in getting results [25]. Furthermore, this model is in general inappropriate in the presence of autonomous participants, because it inherently assumes that providers are homogeneous from a functionality and data point of view and thus provide the same results for queries.

In active query replication, also called *state-machine* [22], both primary and backup providers play the same role: they actively perform queries and, unlike in the passive replication model, there is no centralized control. Active replication does not require checkpointing messages to maintain backup queries and is thus appropriate for distributed systems with autonomous participants. Several solutions have been proposed based on this model. For example, [23] proposes a query allocation algorithm that maximizes the reliability of heterogeneous systems. [10] proposes a scheduling algorithm to achieve fault tolerance in multiprocessor systems. But, these two algorithms can only tolerate a single provider failure, while large distributed systems suffer from many more providers failures. [9] proposes an algorithm using a set of scheduling heuristics that actively replicates each incoming query a fixed number of times, say r, thereby producing schedules that tolerate r providers failure. However, these active query replication solutions replicate each incoming query, which may quickly utilize all computing resources in the system.

Recently, probabilistic approaches have been proposed to deal with failures without replicating each incoming query. For instance, in [1] each processing node and communication link is associated with a failure rate. The authors then tackle the problem of scheduling a task graph with deadline constraints and guaranteeing the best possible reliability. However, this work assumes a constant probability of failure for nodes and considers parallel and homogeneous computers. In [3], authors address the problem of scheduling a set of queries, which are characterized with the same probability of failure, to a set of processors. Given a set of queries and the set of relevant providers, a precise analysis can determine whether replication is required to either guaranteeing high reliability or a minimal set of processors for dealing with a set of queries. However, the authors consider multiprocessor systems and thus make strong assumptions that do not apply to distributed systems with autonomous participants. An advantage of probabilistic approaches though is that, as in our proposal, no assumption on the number of tolerated failures is made. In contrast to our algorithms, these probabilistic solutions assume that providers have the same probability of failure, the same capacity to perform queries, and no intentions at all. None of these assumptions is realistic in large-scale distributed systems.

Our algorithms significantly differ from previous work in four main points. First, to the best our knowledge, this is the first work that uses a probabilistic approach to replicate queries in large-scale distributed systems. Second, in addition to the failure probability of providers, they consider the failure probability of consumers. This consideration is quite important, because repeated consumer failures may cause dissatisfaction of those providers that perform their queries. This is because such providers

waste their computing resources for producing results that are finally not returned to the consumer. Third, our algorithms go further than simply considering failure probabilities: they also consider both participants' satisfaction and queries' criticality to set the query replication rate. This allows our algorithms to only replicate those queries that increase participants' satisfaction. Finally, we consider query replication in its generality: besides replicating queries to tolerate failures, we consider the fact that consumers may also replicate queries to deal with Byzantine providers.

8 Conclusion

In this paper, we focused on active query replication in the presence of autonomous participants. In this context, a way to avoid having several participants to voluntarily leave the system is to satisfy their interests. However, basic query replication techniques can decrease system performance and dissatisfy providers. This is because they quickly overload the system for high query loads and do not consider participants interests at all. Thus, system architects/developers are usually faced to the problem of deciding whether replicating queries is beneficial to the performance of their systems. Usually, system architects/developers decide before-hand how to recover from provider failures: (i) by systematically generating one or more query replicas for any incoming query (the *replicateAll* approach); or (ii) by re-allocating queries after provider failures (the *none* approach). Our goal is to free system architects/developers from all these messy details. For this, we revisited query replication from a satisfaction and probabilistic point of view. To our knowledge, this is the first work that analyzed query replication from a satisfaction point of view.

In summary, we made the following main contributions. First, we extend the satisfaction model presented in [18] to take faulty participants and query criticality into account. Second, we introduced a new global satisfaction notion that characterizes: (i) the criticality of queries for consumers, (ii) the failure probability of participants, and (iv) the satisfaction of participants. Third, we proposed two simple, yet powerful, query replication algorithms to deal with participant failures while considering participants satisfaction, which we called S_bQR and S_bQR+. Both algorithms decide on-the-fly whether a query should be replicated and at which rate such that system performance is not decreased and participants satisfaction is increased. Two salient features of these algorithms is that: they replicate only those queries that allow increasing global satisfaction, and; they make no assumption on how many provider failures might occur at any time. S_bQR+ differs from S_bQR in that it may voluntarily fail to allocate as many replicas as required by consumers for their low critical queries so as to keep computing resources for critical ones.

Our experimental results demonstrated that our algorithms significantly outperform, from the performance and satisfaction points of view, the two most popular baseline branch of algorithms: (i) those that (it systematically generates one replica for any incoming query (*replicateAll*) and (ii) those that only re-allocates queries after provider failures (*none*). The results showed that our algorithms correctly adapt on-the-fly the query replication rate to the criticality of queries and the failure probabilities of participants. This allows our algorithms to ensure good system performance and high participants satisfaction at any point in time. In particular, the results

also show that while replicating systematically all queries suffers from serious performance problems. Our results also showed that while S_bQR is the most adequate to guarantee the number of query instances required by consumers, S_bQR+ is the best choice to increase participants satisfaction and prioritize highly critical queries.

References

1. Assayad, I., Girault, A., Kalla, H.: A Bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In: DSN (2004)
2. Balazinska, M., Balakrishnan, H., Madden, S.R., Stonebraker, M.: Fault-tolerance in the Borealis distributed stream processing system. ACM Trans. Database Syst. **33**(1), 1–44 (2008)
3. Berten, V., Goossens, J., Jeannot, E.: A probabilistic approach for fault tolerant multiproc. Real-time scheduling. In: IPDPS (2006)
4. Bhagwan, R., Savage, S., Voelker, G.M.: Understanding availability. In: IPTPS (2003)
5. Budhiraja, N., Marzullo, K., Schneider, F., Toueg, S.: The primary-backup approach. In: Distributed Systems, pp. 199–216. ACM Press, New York (1993)
6. Castro, M., Costa, M., Rowstron, A.: Performance and dependability of structured peer-to-peer overlays. In: DSN (2004)
7. Chandramouli, B., Bond, C., Babu, S., Yang, J.: Query suspend and resume. In: SIGMOD (2007)
8. Ghosh, S., Melhem, R., Mossé, D.: Fault-tolerant scheduling on a hard real-time multiprocessor system. In: IPDPS (1994)
9. Girault, A., Kalla, H., Sorel, Y.: An active replication scheme that tolerates failures in dist. Embedded real-time systems. In: DSN (2003)
10. Hashimoto, K., Tsuchiya, T., Kikuno, T.: Effective scheduling of duplicated tasks for fault-tolerance in multiprocessor systems. IEICE Trans. Inf. Syst. **E85-D**(3) (2002)
11. Kim, J., Lee, H., Lee, S.: Replicated process allocation for load distribution in fault-tolerant multicomputers. IEEE Comput. **46**(4) (1997)
12. Kossmann, D.: The state of the art in distributed query processing. ACM Comput. Surv. **32**(4), 422–469 (2000)
13. Ni, L., Harwood, A.: A comparative study on peer-to-peer failure rate estimation. In: ICPADS (2007)
14. Pentaris, F., Ioannidis, Y.: Query optimization in distributed networks of autonomous database systems. ACM Trans. Database Syst. **31**(2) (2006)
15. Pieper, S., Paul, J., Schulte, M.: A new era of performance evaluation. IEEE Comput. **40**(9) (2007)
16. Qu, H., Labrinidis, A., Mosse, D.: UNIT: user-centric transaction management in web-database systems. In: ICDE (2006)
17. Quiané-Ruiz, J.A., Lamarre, P., Cazalens, S., Valduriez, P.: Scaling up query allocation in the presence of autonomous participants. In: DASFAA (2011)
18. Quiané-Ruiz, J.A., Lamarre, P., Valduriez, P.: A self-adaptable query allocation framework for distributed information systems. VLDB J. **18**(3), 649–674 (2009)
19. Rahm, E., Marek, R.: Dynamic multi-resource load balancing in parallel DB systems. In: VLDB (1995)
20. Roussopoulos, M., Baker, M.: Practical load balancing for content requests in P2P networks. Distrib. Comput. **18**(6) (2006)
21. Saroiu, S., Gummadi, P.K., Gribble, S.D.: Measuring and analyzing the characteristics of Napters and Gnutella hosts. Multimed. Syst. **9**(2), 170–184 (2003)
22. Schneider, F.: Replication management using the state-machine approach. In: Distributed Systems, pp. 169–197. ACM Press, New York (1993)
23. Shatz, S., Wang, J.P., Goto, M.: Task alloc. for maximizing reliability of dist. com. systems. IEEE Comput. **41**(9) (1992)
24. Wolf, G., et al.: Query processing over incomplete autonomous databases. In: VLDB (2007)
25. Yu, H., Vahdat, A.: The costs and limits of availability for replicated services. In: SOSP (2001)