HAL
open science

Query Replication in Distributed Information Systems with Autonomous Participants

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

To cite this version:

Philippe Lamarre, Jorge-Arnulfo Quiané-Ruiz, Patrick Valduriez. Query Replication in Distributed Information Systems with Autonomous Participants. [Research Report] RR-6928, INRIA. 2009, pp.31. inria-00383321

HAL Id: inria-00383321

https://inria.hal.science/inria-00383321v1

Submitted on 12 May 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Query Replication in Distributed Information Systems with Autonomous Participants

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

N° 6928

April 2009

Thème SYM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6928--FR+ENG

Query Replication in Distributed Information Systems with Autonomous Participants

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

Thème SYM — Systèmes symboliques
Projets ATLAS

Rapport de recherche n° 6928 — April 2009 — 31 pages

Abstract: We consider Distributed Information Systems with Autonomous Participants (DISAP), i.e., participants (consumers and providers) may have special interests towards queries and other participants. Recent applications of DISAP on the Internet have emerged to share data, services, or computing resources at an unprecedented scale (e.g. SETI@home). With autonomous participants, the only way to avoid a participant to voluntarily leave the system is to satisfy its interests when allocating queries. But, participants' satisfaction may also be badly affected by other participants' failures or comportment. In this context, replicating queries is useful to address two different problems: tolerate providers' failures and deal with Byzantine providers. In this paper, we make the following main contributions. First, we formalize the query allocation problem over faulty participants in the context of DISAP. Second, we define participant's satisfaction and define a notion of global satisfaction, which considers participants' satisfaction and their probability of failure. Third, we propose a query replication algorithm, S_bQR, which deals with the participants' failures by deciding on-line whether a query should be replicated and at which rate. Fourth, we propose another query replication algorithm, called S_bQR+, which generalizes S_bQR with the goal of prioritizing critical queries. Finally, we implemented both algorithms and compared them to the popular baseline algorithm. The results demonstrate that our algorithms significantly outperform the baseline algorithm from the performance and satisfaction points of view. In particular, S_bQR+ is excellent at choosing the queries that must be replicated to guarantee both participants' satisfaction and good system performance.

Key-words: Distributed information systems, participants' intentions, autonomous participants, participants' satisfaction, probability of participants' failure, query replication

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Réplication de Requêtes dans les Systèmes d'Information Distribués avec des Participants Autonomes

Résumé : Nous considérons des Systèmes Distribués d'Information dont participants sont autonomes (DISAP, pour ses initiales en anglais), i.e. les participants (consommateurs et fornisseurs) peuvent avoir des intérêts particuliers envers les requêtes et les autres participants. Des applications récentes, sur les DISAP, ont vu le jour dans l'Internet comme pour objectif de partager de données, de services ou de ressources à une très grande échelle (e.g. SETI@home). Avec des participants autonomes, la seule façon d'éviter qu'un participant quitte le système par mécontentement est en satisfaisant ses intérêts au moment d'allouer les requêtes. Cependant, la satisfaction des participants peut être influencée par le comportement ou les pannes des autres participants. Dans ce contexte, la réplication de requêtes est utile pour adresser deux problèmes différents: tolérer les pannes des participants et traîter avec des partipants malicieux (i>e>, Byzantine). Dans cet article, nous faisons les contributions suivantes. Primo, nous formalisons le problème d'allocation de requêtes sur des participants souceptibles de tomber en panne dans le contexte de DISAP. Secondo, nous définons la satisfaction des participants dans les DISAP et définons aussi une notion de satisfaction globle, qui considère al satisfaction et la probabilité de panne des participants. Tertio, nous proposons S_bQR, un algorithme qui tolère les pannes des participants en décidant à la volée si une requête doit être répliquée et combien de fois. Nous proposons aussi S_bQR+ un algortihme de réplication de requêtes, qui généralise S_bQR avec l'objectif de favoriser les requêtes critiques pour les consommateurs. Finalement, nous implementons nos deux algorithmes et les comparons à l'algorithme de base les plus utilisé dans nos jours. Les résultats montrent que nos algorithmes sont beaucoup plus performants dès le point de vue de performance du système ainsi que dès le point de vue de la satisfaction des participants.

Mots-clés : Systèmes d'information distribués, participants autonomes, intentions des participants, satisfaction des participants, probabilité de panne des participants, réplication de requêtes.

1 Introduction

We consider Distributed Information Systems with Autonomous Participants (DISAP) whereby data, services or computing resources can be shared at very large scale. Participants are either resource providers or consumers which submit queries to providers[1]. Examples of DISAP are BOINC [7] and distributed.net [2], which are systems that support the collaboration of high numbers, e.g. millions, of participants over the Internet.

Participants (providers and consumers) are autonomous in the sense that they may leave and join the system at will, but also, they may have special interests (*intentions*) for some queries and other participants. For example, in BOINC and distributed.net, a consumer may want to receive results from highly reputed providers, and a provider may want to perform queries for some preferred projects. In this context, it is crucial to satisfy participants (i.e. to fill their intentions) since dissatisfaction may lead them to leave the system, which may cause some loss of system capacity to perform queries as well as some loss of system functionality. In particular, a participant's departure may yield other participants to leave the system in a domino effect [31].

Because of autonomy, a provider may act maliciously, i.e. may be Byzantine [25], and thus may return wrong or incomplete results for a query. This is why some systems (such as BOINC) allow consumers to replicate queries on different providers so as to compare their results. Queries usually have different importance (which we refer to as *criticality*) for consumers. For instance, it may be crucial for a consumer to receive all its required results for some queries while it may tolerate receiving less results for other queries. Moreover, since participants are normally linked through the Internet, they are subject to network failures. As the scale of the system increases in number of participants, the possibility that one of them fails also increases. Studies of participants' availability in widely deployed distributed systems such as Overnet [11], Napster and Gnutella [34] demonstrate this. Thus, the responsiveness of applications built ontop of DISAP is increasingly limited by providers' availability rather than performance. On the one hand, providers' failures may significantly dissatisfy consumers with no results for their queries. On the other hand, consumers' failures may dissatisfy providers because their results cannot be returned to failed consumers.

Recent solutions have been proposed to deal with different query processing problems in DISAP, e.g., [23, 28, 31, 38, 39], but availability is typically not addressed. A basic solution to deal with providers' failure is to re-allocate, after detection of a provider's failure, the query to another provider. However, this approach may significantly increase response times. Thus, an alternative solution is query replication [8, 9, 10, 19, 20, 40], which can be *passive* or *active*. Passive query replication id based on checkpointing or logging techniques [15, 18], which are not appropriate to DISAP since they inherently assume that providers are using the same algorithms and thus that produce the same results for a query. Furthermore, this may significantly increase response times because there is a system overhead for detecting a provider's failure, determining which queries have been stopped, and rescheduling stopped queries. Active query replication is more adequate to DISAP. It allocates queries to the number of providers required by the consumer (called primary providers) plus some other providers (called backup providers), so that results produced by backup

[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.

providers are returned to consumers in case of primary provider's failure. We henceforth refer to those queries allocated to backup providers as backup queries. In our context, we observe that active query replication is useful to deal with both Byzantine providers and providers' failures. Usually, replicating queries to deal with Byzantine providers is left to consumers [7, 16], while it is up to the system to replicate queries to deal with providers' failures because it may better know participants' failure. None of existing query replication solutions considers autonomous participants.

In this paper, we consider active query replication in DISAP, from a satisfaction point of view, to tolerate participants' failures. Supporting query replication in DISAP is challenging for several reasons. First, the overhead of query replication may outweigh its benefits, by over-utilizing computing resources or requiring either more powerful providers or additional providers. Second, a provider may not have the same intention, and thus the same satisfaction, of being utilized as primary provider or backup provider. This is because the query results produced by a backup provider are only returned to the consumer in case the primary providers fails, which means that it may consume its computing resources for nothing. Third, providers may also consume their computing resources for nothing in case of consumers' failure. In the rest of this section, we first illustrate typical applications of DISAP, then provide a motivating example, and finally introduce our main contributions.

1.1 Applications

Applications of different domains need to deal, in an automated way, with participants' failures in order to operate correctly. *Volunteer computing*, *Web services*, *cloud computing*, and *Grid computing* are some examples.

VOLUNTEER COMPUTING. It involves a distributed computing system where computer's owners (the providers) donate, in a transparent, open, and scalable way, their services or resources to one or more projects (the consumers). BOINC [7], XGrid [5], and Grid MP [6] are examples of such systems. SETI@home [22] and grid.org [3] are examples of scientific applications running on one of these systems. An important requirement in these applications is that the system should be able to deal with participants' failures.

WEB SERVICES. Systems based on Web services [4] can also be DISAP. Web services are rapidly becoming a standard way of communication among loosely-coupled, heterogeneous systems. Recently, [37] proposed the overall goal of a Web Service Management System (WSMS) that allows consumers to query a collection of Web services (the providers) simultaneously in a transparent and integrated way. The authors focus on the query optimization issues that arise in a WSMS in order to speed up query execution. They assume that Web services do not fail when they run queries, which is not the case in practice. Thus, query replication can be used by a WSMS to ensure results for consumers. Nevertheless, because of Web services autonomy and load management, queries cannot be replicated in a systematic way.

CLOUD COMPUTING. This paradigm refers to the use of memory and capacity of personal computers and servers around the world linked by a network. Amazon S3 [1] and Google File System [17] are examples of available "clouds". Cloud's users (the consumers) can then make use of a considerable and scalable computing power (the providers aggregation), but also of providers' services, data, and storage capacity. Cloud computing has three main features: (i) it is query-intensive,

(ii) its network topology is highly dynamic, and participants may join, leave, or fail over time, and (iii) it should serve queries with short response times. Because of these features, it is crucial that providers' failures be handled in a preventive way so that the query load be not significantly increased.

GRID COMPUTING. It is a general form of parallel computing whereby several networked, loosely-coupled, and heterogeneous computer nodes function as one large "supercomputer". It has been used by research labs or companies to mutualize their distributed computing resources and parallelize the processing of heavy loads of queries or very large queries. The geographical dispersion and potentially high number of grid nodes makes the probability of a node failure quite high and thus makes it important to support node failures in a way that does not loose queries or subqueries.

The solution we propose in this paper can be applied in these domains to dynamically decide if a query must be replicated and at which rate depending on query's criticality, participants' probability of failure, and participants' satisfaction.

1.2 Motivating Example

Let us illustrate query replication in DISAP with an application from volunteer computing using BOINC (*Berkeley Open Infrastructure for Network Computing*). BOINC allows scientific communities to create and operate public-resource computing applications by using computing resources of thousands of volunteers across the world. The query processing principle in BOINC is the following. Applications (the consumers) submit their queries to BOINC to be executed 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 a simple scenario where a given research project running on BOINC (i.e., a consumer in the BOINC system) sends a query, with a very high criticality, requiring results from 1 provider. Suppose that, when the query arrives in the system, the relevant providers (those which can treat the query) have low workload, high intentions to perform the incoming query, and high failure probability, which raises the probability of restarting the allocation of the query or to have a dissatisfied consumer. 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 10 providers. Suppose that, when the query comes in, the relevant providers have high workload, low failure probability, and low intentions to perform the incoming query. In this case, it is better not to replicate the query to neither overload nor dissatisfy providers. Furthermore, not replicating this query allows to give higher priority to queries with high criticality.

However, most cases are not so simple and raise the following questions: *which queries should be replicated?* and *how many query instances should be replicated?* Then, a preliminary question is: *which information must be used to decide on query replication? consumers' satisfaction? providers' satisfaction? queries' criticality? others?*. This paper answers those questions.

1.3 Contributions and Outline

After surveying related work in Section 2, we present our main contributions. First, in Section 3, we formally define the query allocation problem over faulty participants. To this end, we first formalize the query allocation problem in DISAP. Second, we define in Section 4 a satisfaction model that considers participants' failures. In particular, we 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 because of backup queries and consumers' failures. Third, in Section 5, we propose two autonomic query replication algorithms: S_bQR and S_bQR+. Both algorithms consider queries' criticality, participants' satisfaction, and participants' failure probability to decide on-line which queries should be replicated and how many backups should be created. S_bQR+ complements S_bQR with the ability to prioritize queries with high criticality. To do so, it may voluntarily reduce the number of required providers by a consumer. Fourth, in Section 6, we experimentally show that our algorithms perform better than the popular baseline algorithm. Overall, we demonstrate our algorithms' self-adaptability to the workload, queries' criticality, and participants' failure probabilities. We also demonstrate that systematic replication of all incoming queries causes serious performance problems for high workloads, and worse, that it loses more query results than with no replication. We show that neither S_bQR nor S_bQR+ have this problem, which allows a system to scale up. Finally, Section 7 concludes.

2 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 [13], primary providers actively perform queries and regularly checkpoint their state to backup providers [9], 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 [40]. Furthermore, this model is inappropriate for DISAP since 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* [35], 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 to DISAP. Several solutions have been proposed based on this model. For example, [36] proposes a query allocation algorithm that maximizes the reliability of heterogeneous systems. [20] proposes a scheduling algorithm to achieve fault tolerance in multiprocessor systems. But, these two algorithms can only tolerate a single provider's failure, while DISAP may suffer from many more providers' failures. [19] 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 replica-

tion 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 [8] 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 [10], 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 open distributed systems, such as DISAP. 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, such as DISAP.

Our algorithms significantly differ from previous work in fourth 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 in DISAP 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 replication rate of queries. 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.

3 Problem Definition

In this section, we first give a presentation of the system model which characterizes DISAP and define the query allocation problem. Then, we formally define the problem of query allocation over faulty participants.

3.1 System Model

We adopt the usual architecture of a mediator m, and of a set $\mathcal{I}$ of autonomous participants. Participants may play two different roles: consumer and provider. The set of consumers and providers are denoted by C ($C \subseteq \mathcal{I}$) and P ($P \subseteq \mathcal{I}$), respectively. Besides autonomy, we assume that participants perform queries when they are required for. In other words, providers must answer queries as much

as they can: they can fail, but only for network failure or a software dysfunction. To formalize this aspect, we assume that each participant $i \in \mathcal{I}$ has a probability f_i to fail per time unit. The way in which participants' failure probability is computed is well beyond the scope of this paper. We simply assume that the mediator is able to estimate this probability by applying for example the solutions in [14, 27]. Mediator m may also fail but this is orthogonal to the focus of this paper. Replicating mediators is a solution to deal with mediator's failure but, in this paper, we simply assume that m never fails.

Providers in P are potentially heterogeneous in terms of capability, capacity, and data. 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. The *capacity* of a provider $p \in P$, $cap(p)$, denotes the number of computing units (e.g. expressed in time units) that it has to perform queries. Thus, p's *utilization* at time t, $\mathcal{U}_t(p)$, is defined as its load w.r.t. its capacity. Finally, data heterogeneity means that providers may produce different results for a same query.

A consumer $c \in C$ submits a query to mediator m when it cannot locally perform the query or because it wants to outsource the query. A consumer formulates queries in a format abstracted as a 4-tuple $q = < c, d, n, \gamma >$. 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, $q.n \in N^*$ is the number of providers to which the consumer wishes to see its query be allocated, and $q.\gamma \in [0..1]$ denotes the criticality of the query. The greater the value of this parameter is, the more critical the query is. Notice that, a consumer may desire to allocate a query q to various providers ($q.n > 1$) for two main reasons: *(i)* to avoid Byzantine providers by comparing their results, and *(ii)* to select the best query result since providers may produce different results for a same query. In the following, 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 $q.c$ (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. 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. We do not make any restriction on the way participants compute their intentions and only assume they provide them in the interval $[-1..1]$. 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. Notice that, participants' failures may dissatisfy other participants with no results for their queries or with results that are not returned to consumers. This means that providers utilize their computing resources for nothing. This is why providers also express the cost of performing a query q and that consumers give the criticality of their queries so that the mediator strives to ensure results for their critical queries. Providers' costs are in the interval $[0.. + \infty[$ and stored by m in vector $\overrightarrow{PC}_q$. For example, given an incoming query q, a cost $\overrightarrow{PC}_q[p] = 100$ could mean the number of milliseconds that provider p needs to perform query q. Finally, the way in which mediator m obtains this information strongly depends on the system architecture. In some cases, it may locally have all this information available, as in cluster-based systems, and in other cases it must ask participants for

this information, as in DISAP. In fact, in DISAP, participants are the only ones that can provide their intentions for each incoming query since such intentions depend on participants' precise context.

Having stated the system model, we can stress that a satisfactory query allocation is one that not only strives to ensure the number of results desired by consumers, but also, one that selects the most interesting providers for consumers and allocates the most interesting queries to providers. We formalize this in the following section.

3.2 General Query Allocation Problem

We can now state the query allocation problem as follows. Given a set $\mathcal{I}$ of autonomous participants, mediator m must 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 it into the three following independent subproblems.

FINDING RELEVANT PROVIDERS. To find those providers that can deal with a query q (the relevant providers), denoted by set P_q ($P_q \subseteq P$), is a matchmaking problem. In our context, for any incoming query q, parameter $q.d$ is intended to be used within a matchmaking mechanism to find set P_q. This problem has been extensively studied in the literature and several matchmaking mechanisms have been proposed [24, 26]. Such techniques are well beyond the scope of this paper and we simply assume there exists one in the system that is sound and complete.

RANKING RELEVANT PROVIDERS. To allocate a given incoming query q, the mediator usually considers a specific strategy. For example, the mediator may score providers by considering: providers' utilization for applications requiring to query load balancing [32, 33]; provider's relevance to queries for web search applications [12]; or participants' intentions for volunteer computing applications such as BOINC and Xgrid [31]. Thus, it could exist as many scoring functions as types of applications. This problem is also out the scope of this paper and we assume that the mediator can provide a vector $\overrightarrow{R}_q$ of ranked providers, according to its strategy, so that $\overrightarrow{R}_q[1]$ is the best scored provider and $\overrightarrow{R}_q[||P_q||]$ is the worst scored provider. The way in which vector $\overrightarrow{R}$ is computed is a choice of the system administrator depending on the system's challenges she wants to solve.

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., $\widehat{P_q^r} = \overrightarrow{R}_q[1..r]$. As the providers' ranking process, the providers' selection process is key to the well operation of the system. A natural solution to this subproblem 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. Moreover, it is possible that, instead of set $\widehat{P_q^r}$, 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. Thus, the mediator should set r by considering this aspect.

3.3 Query Allocation over Faulty Participants

In this paper, we address the problem of *selecting relevant providers* (the third subproblem of the previous section) in DISAP, i.e. with faulty participants. We formalize this problem as follows.

PROBLEM STATEMENT. Given an incoming query q and vector $\overrightarrow{R}_q$ of providers, each p in vector $\overrightarrow{R}_q$ 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 with the query allocation.

The difficulty of addressing this problem is in determining on-line the r value so that consumers receive in general their queries' results from the required number of providers and are satisfied. For low workloads, it is easy to address this problem since there are enough computing resources to perform queries, including backup queries, and thus one can replicate queries without fearing the consequences (except if such replication hurts participants' satisfaction). Nevertheless, the above problem's difficulty increases as the workload gets higher. This is because the load due to backup queries may induce even more significant problems than initial queries. In particular, replicating queries for high workloads impacts performance with longer response times, which increases the probability that a provider fails before performing a query.

A systematic solution to the above problem is to allocate q to the $q.n+m$ best ranked providers [21]. However, this solution only supports the failures of m providers. Moreover, allocating each query to a high number of providers may quickly overload the system as well as decrease participants' satisfaction. Probabilistic approaches have been proposed to define at run time whether a query must be replicated [8, 10]. Nevertheless, they do not consider participants' satisfaction nor consumers' failures, and assume the same providers' failure probability. Therefore, replicating queries in DISAP is hard because one needs to ensure good system performance, high participants' satisfaction, and results for queries with high criticality. To our knowledge, tolerating participants' failures is still an open issue in DISAP and has not been addressed before.

4 Satisfaction Model

In this section, we propose participants' satisfaction definitions that consider query's criticality and the possibility of failure (Sections 4.1 and 4.2). We also propose a definition of global satisfaction, which exploits the failure probability of participants (Section 4.3). We consider the satisfaction of a participant w.r.t. the way in which queries are allocated by the mediator. It is clear that the term "satisfaction" can have a much broader interpretation and may be linked to many other points, for instance, the quality of results. Indeed, it is possible for a consumer to be satisfied of the way in which queries are allocated (just because the allocation process follows its recommendations), but dissatisfied of the obtained results. Thus, "satisfaction" has many different facets and exploring all the ways to compute satisfaction is well beyond the scope of this paper. However, anticipating Section 5, the algorithms we propose are quite general and can be used with any satisfaction definition.

4.1 Consumer's 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). [30] 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. [31] 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 [31], 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,*

$$\Delta_q^x = \left| \begin{array}{ll} \dfrac{1-\gamma}{1-\gamma \cdot \frac{x}{n}} & if\ \gamma < 1 \\ 1 & if\ \gamma = 1 \wedge x = n \\ 0 & if\ \gamma = 1 \wedge x \neq n \end{array} \right.$$

It is worth noting in Equation 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 Figure 1. Observe that as the query's 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 consumer's intentions average concerning the set of providers that return results multiplied by 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,*

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

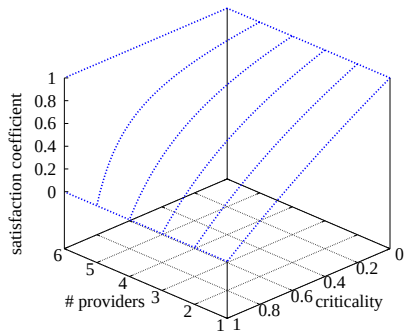


Figure 1: Number of providers returning results versus query's criticality, when a consumer requires 6 providers.

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.

4.2 Provider's Satisfaction

A provider that has not failed can evaluate, by means of its satisfaction, if the mediator allocates queries according to its intentions[2]. 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, and (3) to perform a query as backup for nothing. This is because a provider is usually selfish and hence the fact of spending computing resources to perform queries from which it obtains no benefit does not meet its intentions at all. A relevant provider may have one of three possible states after the allocation of a given query, which is formally stated in Definition 3.

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,*

$$\delta_s(p, \widehat{P_q^r}, \widehat{\widehat{P_q}}) = \left| \begin{array}{ll} \big(\overrightarrow{PI}_q[p] + 1\big)/2 & if\ p \in \widehat{\widehat{P_q}} \\ \big(-\overrightarrow{PI}_q[p] + 1\big)/2 & if\ p \in (P_q \backslash \widehat{P_q^r}) \cap P_q^{ok} \\ 1/\big(2 + \overrightarrow{PC}_q[p]\big) & if\ p \in \big(\widehat{P_q^r} \backslash \widehat{\widehat{P_q}}\big) \cap P_q^{ok} \end{array} \right.$$

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

[2] A provider which fails simply does not compute its satisfaction.

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 of a provider.

4.3 Global Satisfaction

According to the satisfaction definitions of the previous sections, query replication can improve consumer's satisfaction, which is the goal of replicating queries and is a positive aspect. However, 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. 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 in average. Nevertheless, participants usually compute their satisfaction after query allocations, or even after receiving results, while decisions to replicate queries are done before allocating queries. Thus, we define a global satisfaction notion that takes place before query allocations. Since this global satisfaction notion is computed before query allocations, it depends on the participants' failure. Hence, we must consider all the possible cases of failure, which requires some work in probabilities. With this aim, we first characterize, in Section 4.3.1, the successful probability that: (i) a query be treated by some number of providers, and (ii) the results produced by a provider be returned to consumers. We then define global satisfaction in Section 4.3.2.

4.3.1 Probabilities of success

We assume that faults are not correlated. Thus, the probability that a participant i does not fail in a time unit is $1 - f_i$, f_i being 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 $\mathcal{A}_q^i$ that i does not fail in a discrete time interval t_q^p (i.e. to be always available during time interval t_q^p) is given by Equation 1.

$$\mathcal{A}_q^i = (1 - f_i)^{t_q^p} \tag{1}$$

Given this, the probability that at most h providers in set $\widehat{P_q^r}$ do not fail before returning results of a query q is given by,

$$\mathcal{S}_q^h(\widehat{P_q^r}) = \sum_{\substack{P_q^{ok} \subseteq \widehat{P_q^r} \\ ||P_q^{ok}|| \leq h}} \Big(\prod_{p \in P_q^{ok}} \mathcal{A}_q^p \prod_{p \in \widehat{P_q^r} \setminus P_q^{ok}} (1 - \mathcal{A}_q^p) \Big) \tag{2}$$

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

$$\mathcal{S}_q^a(\widehat{P_q^r}, x) = \left| \begin{array}{l} \displaystyle\sum_{\substack{\widehat{\widehat{P_q}} \subseteq \widehat{P_q^r} \\ ||\widehat{\widehat{P_q}}||=x \\ \overrightarrow{R}_q[a] \in \widehat{\widehat{P_q}}}} \Big(\prod_{p \in \widehat{\widehat{P_q}}} \mathcal{A}_q^p \prod_{p \in \widehat{P_q^r} \setminus \widehat{\widehat{P_q}}} (1 - \mathcal{A}_q^p) \Big) \; if \; x < q.n \\ \displaystyle\sum_{\substack{\widehat{\widehat{P_q}} \subseteq \widehat{P_q^r} \\ ||\widehat{\widehat{P_q}}||=x \\ \overrightarrow{R}_q[a] \in \widehat{\widehat{P_q}}}} \Big(\prod_{p \in \widehat{\widehat{P_q}}} \mathcal{A}_q^p \prod_{\substack{p = \overrightarrow{R}_q[j] \\ j \leq \max(k) \\ \overrightarrow{R}_q[k] \in \widehat{\widehat{P_q}} \\ p \notin \widehat{\widehat{P_q}}}} (1 - \mathcal{A}_q^p) \Big) \quad else \end{array} \right. \tag{3}$$

4.3.2 Global satisfaction definition

We can then define global satisfaction with respect to the allocation of a given query. Informally, 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, the global satisfaction denotes the sum of the relevant providers' satisfactions plus the consumer's satisfaction. Unfortunately, possible participants' failure make this a little more complicated. For a relevant provider that does not fail, we must consider the three cases defined in Section 4.2. The first case is when a provider is allocated a query and its results are returned to the consumer. For this to happen, it is necessary that the consumer does not fail and that no more than $n-1$ (where n is the number of required providers) best ranked providers do not fail and hence return results. In this case, the provider's satisfaction is based on its intention. Except in case that the consumer fails: the provider's satisfaction is based on the query's cost. The second case is when a provider is allocated a query replica and its results are not returned to the consumer. This happens if at least n best ranked providers do not fail. In this case, the provider's satisfaction is based on the query's cost. Finally, the third and last case is when a provider is not allocated a query. Here, the provider's satisfaction is based on its negative intention. For the consumer, we simply need to consider the probability that each relevant provider has to return results. Definition 4 formalizes the global satisfaction's computation.

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,*

$$\begin{aligned}
\Theta(\widehat{P_q^r}) \quad = \quad & \sum_{j=1}^{r} \Bigg(\mathcal{A}_q^{\overrightarrow{R}_q[j]} \cdot \Big(\mathcal{A}_q^c \cdot \mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}}) \cdot \overrightarrow{PI}_q[\overrightarrow{R}_q[j]] \quad + \\
& \qquad \big(1 - \mathcal{A}_q^c\big) \cdot \mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}}) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[j]] + \\
& \qquad \big(1 - \mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}})\big) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[j]] \Big) \Bigg) \quad + \\
& \sum_{j=r+1}^{||P_q||} \mathcal{A}_q^{\overrightarrow{R}_q[j]} \cdot -\overrightarrow{PI}_q[\overrightarrow{R}_q[j]] \quad + \\
& \mathcal{A}_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \sum_{a=1}^{r} \Big(\mathcal{S}_q^a(\widehat{P_q^r}, j) \ \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[a]] \Big) \right)
\end{aligned}$$

5 Algorithms

In this section, we describe two new algorithms to perform query replication in DISAP. The first algorithm, called S_bQR, implements a typical query replication strategy. That is, it aims at creating some query instances in addition to those asked by consumers so that consumers receive queries' 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, it prioritizes highly critical queries by creating less queries instances for low critical queries, than required by consumers. Both algorithms respect the strategy of the mediator to allocate queries and compare the global satisfaction of different provider sets in order to select the best providers. We give some considerations of this global satisfaction's comparison in Section 5.3.

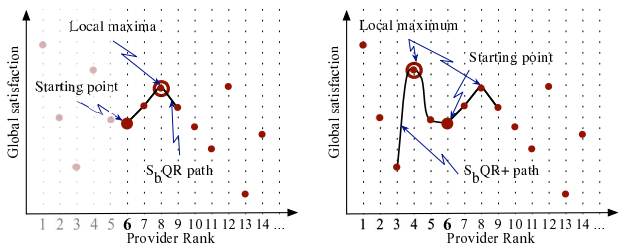
5.1 S$_b$QR Algorithm

Satisfaction-based Query Replication (S_bQR for short) is an algorithm to replicate queries in order to support participant failures in DISAP. 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 incoming query. The basic idea is simple: it creates as many backup providers as long as global satisfaction increases. In more details, given an incoming 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 $\overrightarrow{R}_q[1..n+i]$ of providers whose global satisfaction is greater than $\overrightarrow{R}_q[1..n+i+1]$. In other words, S_bQR looks for the local maximum of global satisfaction.

Algorithm 1: S_bQR

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

```
begin
    r = q.n;
    while r < R_q.size && Θ(P̂_q^r) < Θ(P̂_q^{r+1}) do r++; return P̂_q^r;
end
```



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

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

EXAMPLE. We illustrate in Figure 2(a) the S_bQR's principle 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

S_bQR+ is an algorithm that addresses the problem stated in Section 3.3 from a more general point of view. That is, it not only analyzes if backup queries must be created, 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 global satisfaction. This is quite useful for heavy workloads when replicating queries may significantly hurt system performance. But, S_bQR+ always allocates at least one query instance so that queries be treated (i.e. $q.n \geq 1$). Thus, it could be a good idea to allocate low critical queries to less providers than those required so as to keep computing resources for highly critical queries. To do so, S_bQR+ looks for the local maximum of global satisfaction by also analyzing global satisfaction

Algorithm 2: S_bQR+

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

```
begin
    P̂_q^+ = S_bQR(q, R⃗_q, CI⃗_q, PI⃗_q, PC⃗_q)
    r = q.n;
    while r > 1 && Θ(P̂_q^r) < Θ(P̂_q^{r-1}) do r--; P̂_q^- = P̂_q^r;
    if Θ(P̂_q^-) < Θ(P̂_q^+) then return P̂_q^+;
    else return P̂_q^-;
end
```

Theorem 1

$$
\begin{aligned}
\Theta(\widehat{P_q^{r+1}}) - \Theta(\widehat{P_q^r}) \quad = \quad & \mathcal{A}_q^{\overrightarrow{R}_q[r+1]} \cdot \Big(\mathcal{A}_q^c \cdot \mathcal{S}_q^{n-1}(\widehat{P_q^r}) \cdot \overrightarrow{PI}_q[\overrightarrow{R}_q[r+1]] \quad + \\
& (1-\mathcal{A}_q^c) \cdot \mathcal{S}_q^{n-1}(\widehat{P_q^r}) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[r+1]] \quad + \\
& \big(1-\mathcal{S}_q^{n-1}(\widehat{P_q^r})\big) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[r+1]] \quad + \\
& \overrightarrow{PI}_q[\overrightarrow{R}_q[r+1]] \Big) \quad + \\
& \mathcal{A}_q^c \cdot \sum_{j=0}^{n} \Bigg(\Delta_q^j \cdot \frac{1}{n} \cdot \Big(\sum_{a=1}^{r} \big(\mathcal{S}_q^a(\widehat{P_q^{r+1}}, j) - \mathcal{S}_q^a(\widehat{P_q^r}, j)\big) \cdot \overrightarrow{CI}_q[a] \quad + \\
& \mathcal{S}_q^{r+1}(\widehat{P_q^{r+1}}, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[r+1]] \Big) \Bigg)
\end{aligned}
$$

when reducing queries' instances. We illustrate the S_bQR+ process in Algorithm 2. S_bQR+ has the following two properties: (i) it never creates more backup queries than S_bQR, and (ii) 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 Definition 2, the consumer's satisfaction (regarding the mediator's job) 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 a query's instances. But also, a consumer may be satisfied, with the received results, even if it is not the desired number.

EXAMPLE. To exemplify the S_bQR+'s principle 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 ($\gamma = 0.1$). Since, 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 Figure 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 save computing resources, which could be devoted to highly critical queries.

5.3 Global Satisfaction Computation

Both 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 providers' set has the highest global satisfaction, one should compute the global satisfaction of both sets. One could be afraid by such a comparison because it is complicated and it may be long to realize. However, it is possible to take advantage of the fact that providers' sets are built according to the query allocation strategy (i.e., using vector $\overrightarrow{R}$): the difference among two compared sets of providers is always one and only one provider. Thus, global satisfaction comparison can be reduced to the study of the impact of adding a provider from a given set of providers, which significantly simplifies the computation. We formalize this analysis in Theorem 1. See the proof of this theorem in Appendix A. Anticipating the validation section, we experimentally analyze this global satisfaction comparison and see that its required time is 50 milliseconds, which is negligible.

5.4 Discussion

We pointed out in Section 4 that there may exist several definitions of satisfaction and that one may see a satisfaction based on the quality of results as an intuitive definition. However, in our context, it is difficult to include satisfaction with respect to answers since *(i)* participants' evaluations of a particular answer are private data that the mediation process does not have access to, and *(ii)* by composition, our proposal takes place before answers computation (before query allocations) and hence we do not have any information about the results produced. Our algorithms are quite general and independent of the way in which satisfaction is computed. Thus, one can adapt the satisfaction definition to fit a particular application. This also applies to participants' intentions, where participants may consider any information they have, such as personal experiments, participants' reputation, response time, and load. Nonetheless, it is worth noting that the system's behavior strongly depend on the way participants compute their intentions. For instance, if participants do not care about their preferences and compute their intentions by only considering providers' load, as a result one will have a system that ensures short response times.

Moreover, the scoring function (which produces vector $\overrightarrow{R}$) is usually based on specific demands, which are given by the application challenges that one wants to solve. Thus, a large number of specific query allocation methods with different behaviors may exist. For example, the score function of a *qlb* method is designed for those applications whose goal is to ensure good system performance. However, our algorithms treat the ranking function a black box, which allows them to preserve the mediator's strategy for allocating queries by preserving the ranking order of providers when finally

Parameter	Definition	Value
nbConsumers	Number of consumers	150
nbProviders	Number of providers	300
nbMediators	Number of mediators	1
qDistribution	Query arrival distribution	Poisson
iniSatisfaction	Initial satisfaction	0.5
γ	Query criticality	from 0.3 to 1
fRate	Participant failure rate	0.03/second
nbRepeat	Repetition of simulations	10

Table 1: Simulation parameters.

deciding which providers are allocated a query. Therefore, our approach can be applied in any kind of application.

Finally, notice that the complexity of our algorithms is $\Theta(n \cdot r^2)$, but with a quite small n and r (usually $n < 10$ and $n \simeq r$).

6 Experimental Study

In this section, we validate our algorithms by comparing them with a popular baseline algortihm, $replicateAll$. $replicateAll$ allocates each incoming query to a single backup provider, no matter how many providers a consumer desires [21]. In other words, $replicateAll$ allocates an incoming query q to $q.n + 1$ providers. To clearly see our algorithms gains, we also study the case where no backup query is generated (we call it the $none$ case). We carry out our experimental validation with three main objectives: (i) to evaluate how well, from a satisfaction point of view, our algorithms operate in DISAP; (ii) to evaluate the impact on performance of backup queries generated by S_bQR; and (iii) to analyze if our algorithms can adapt to different query criticitie and to different probabilities of participant failures.

6.1 Setup

We implemented our prototype in Java. To compute vector $\overrightarrow{R}$, we first implemented S_bQA [31] and then implemented S_bQR, S_bQR+, and $replicateAll$ algorithms on top of S_bQA. For all the query allocation methods we tested, the configuration (Table 1) is the same and the only change is the way in which each query replication algorithm creates backup providers. Before defining our experimental setup, let us point out that the definition of a synthetic workload for environments where participants are autonomous and have special interests towards queries is an open problem. In [29] 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. Another possibility to validate our results would be to consider real-world data over long periods of time. However, even if we had the resources to obtain real-world data, the validation would get biased towards these specific applications. Therefore,

in our experiments, we decided to generate a very general workload that can be applied to different applications and environments in order to thoroughly validate our results.

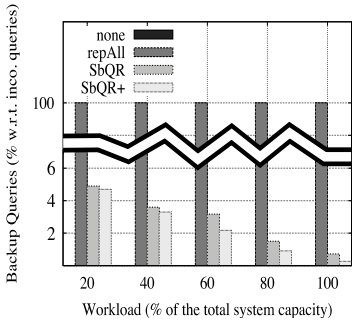
We generated a network with 150 consumers and 300 providers who compute their satisfaction as presented in Sections 4.1 and 4.2, respectively. We consider a single mediator. We initialize participants' satisfaction with a value of 0.5. Then, participants make an average of their satisfaction over the last 150 issued queries (for consumers) and the 400 queries that have passed through providers. We assume that all 300 providers are able to perform any incoming query issued by consumers. We assume that a participant has a probability of 0.03 of failure per second. This is a high failure probability, but we want to stress the system in order to conduct our experimental study under difficult situations. Following the approach used in [34], we generate around 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 divide 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).

The way participants compute their intentions and utilization is beyond the scope of this paper and orthogonal to the problem addressed here. Thus, without loss of generality, we assume that participants work their intentions out as defined in [31]. Let us recall, on the one hand, that providers consider their preferences, utilization, and satisfaction to compute their intentions, while consumers only consider their preferences. We assume that participants compute their preferences uniformly at random between -1 and 1. We assume that it is up to a provider p to estimate the time it needs to perform each incoming query q and gives it to the mediator. We assume that all participants have the same network capacities. We run 10 experiments for each case to present the average results of all experimentations. We produce, in average, 15000 incoming queries for a series of experiments. We generate two classes of queries that *high*-capacity providers perform in 1.3 and 1.5 seconds, respectively, and assume that they arrive in a *Poisson* distribution. Consumers issue queries with a criticality that they generate at random between 0.3 and 1. We assume that consumers ask results for six different providers (i.e., $n = 6$). Since our goal is to study how our algorithms replicate queries, we assume captive participants in these experiments so that participant departures by dissatisfaction do not impact the results. In the following, we always present results for different workloads, where the workload is with respect to the number of queries issued by consumers and any backup query generated by the algorithms we test is added to such workload.

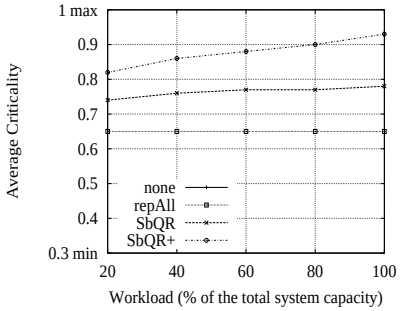
6.2 Results

We can study many different aspects of our algorithms. However, because of space limitations, we focus on only 6 major aspects: the number of created backup queries (Figure 3(a)), the average criticality of backup queries (Figure 3(b)), the number of missing results (Figure 3(c)), the average criticality of queries with missing results (Figure 3(d)), the ensured response times (Figure 3(e)), and the consumer's satisfaction (Figure 3(f)).

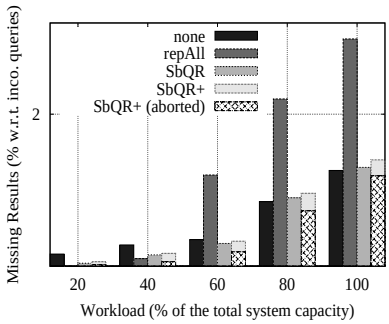
In Figure 3(c), we can observe that, for low workloads, $replicateAll$ has (around 0.01%) less queries with missing results than the other algorithms. However, we can observe in Figure 3(a)



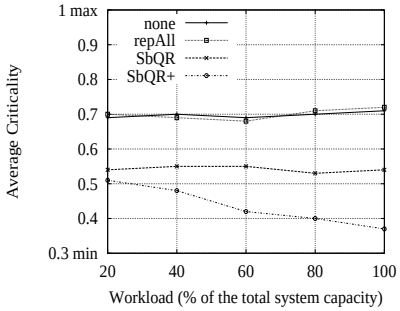
(a) Backup queries created for different workloads.



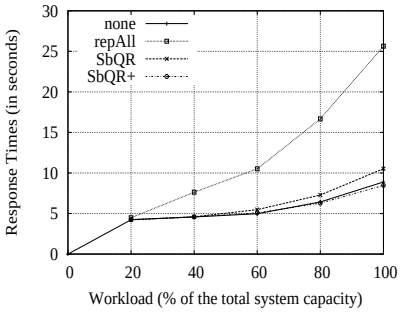
(b) Average criticality of backup queries.



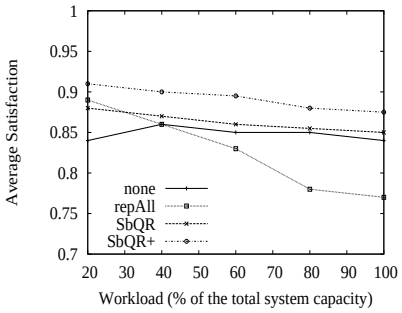
(c) Missing Results for different workloads.



(d) Average criticality of queries with a missing result.



(e) Response times.



(f) Consumers' satisfaction mean for different workloads.

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

that it replicates 20 times more queries. The disadvantages of $replicateAll$ becomes clear in Fig-

ure 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. Figure 3(a) allows to clearly see that they create much less backup queries as the workload increases. Furthermore, both S_bQR and S_bQR+ consider the criticality of queries by mainly replicating queries whose criticality is greater than the average criticality of incoming queries (see Figure 3(b)). This trend is more important as the workload is higher, in particular for S_bQR+. The advantage is that the number of missing results is similar to the ones of the $none$ case and ensures more results for highly critical queries (see Figure 3(d)).

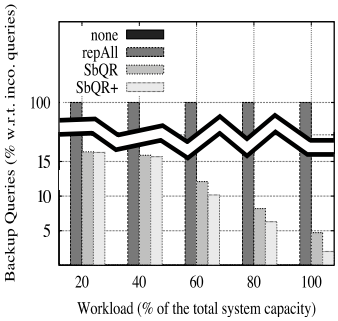
It is worth noting that, in the S_bQR+ case, the number of aborted queries, i.e. those which were allocated to less providers than those required, increases with the workload. S_bQR+ voluntarily aborts lowly critical queries (see Figure 3(d)) to prioritize highly critical queries (see Figure 3(b)). As a result, the number of queries with missing results (due to providers' failure) also increases, but much more slowly (see Figure 3(c)). Figure 3(f) clearly shows that consumers appreciate this. Furthermore, aborting lowly critical queries allows S_bQR+ to guarantee short response times (see Figure 3(e)). Notice that, even if S_bQR has slightly longer response times than when doing no replication (the $none$ case), S_bQR also significantly outperforms $replicateAll$. During our experiments, we observed that the required time to compare the global satisfaction of two sets of providers is in average 50 milliseconds. We also 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 have problems with workloads higher than 40% while our algorithms remain stable.

In summary, the results demonstrate that our algorithms allow ensuring participants' satisfaction, results for critical queries, and short response times at very low cost, in numbers of backup queries and computing cost. Furthermore, our algorithms are self-adaptable to the workload as well as to queries' criticality, participants' intentions, and participants' failure probability. In the following, we go further with our validation with the aim of evaluating how well S_bQR and S_bQR+ (i) operate in environments where participants have no preference, (ii) deal with different queries criticality, and (iii) deal with high probabilities of participants' failure.

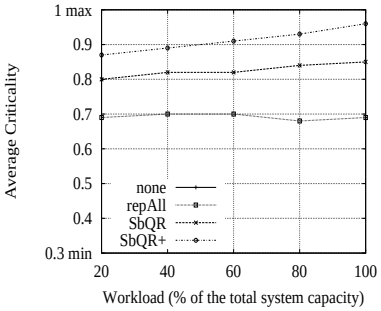
6.2.1 Results with passive participants

One may wonder whether the previous results are impacted by participants' preferences. To address this question, we ran again the series of experiments of previous section, but now assuming that the participants act as nodes in a cluster[3]. This means that consumers are not able to make any difference between two providers and that providers do not have any preference on what they do. Technically, in our context, this means that participants' preference is equal to 1. In other words, consumers always express intentions equal to 1 and providers only consider their utilization to compute their intentions. We also assume that backup providers express a query cost of 0. This means that providers are completely at the system's disposal. Notice that results for the $none$ and $replicateAll$ cases are the same as in the previous section since they do not consider participants' intentions to operate.

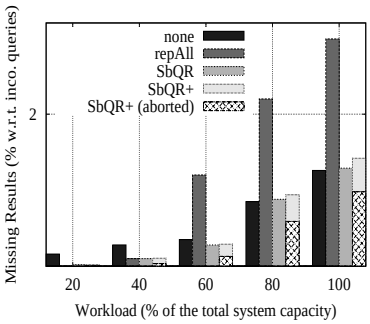
[3]This assumption only holds for this section.



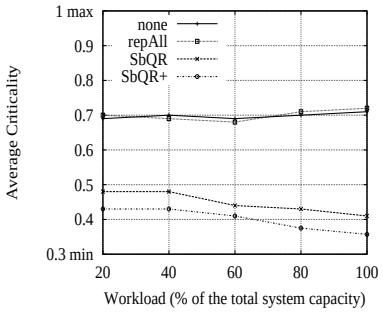
(a) Backup queries created for different workloads.



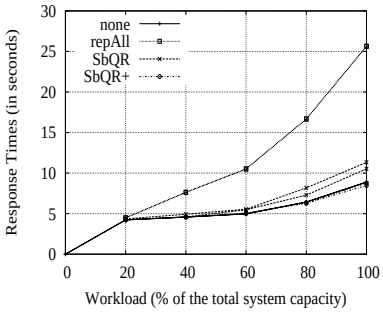
(b) Average criticality of backup queries.



(c) Missing Results for different workloads.



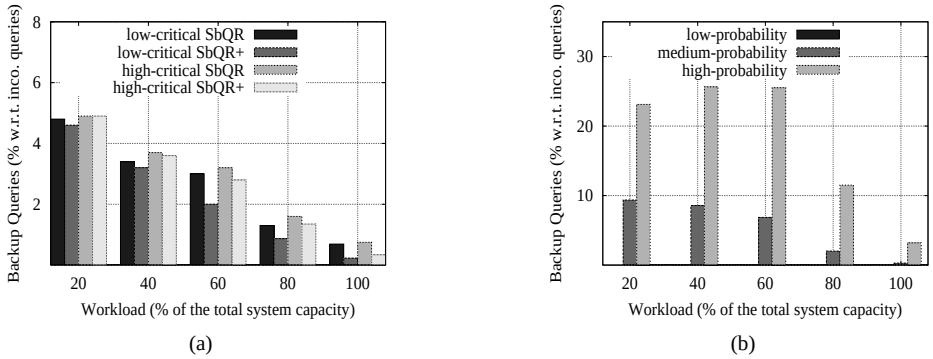
(d) Average criticality of queries with a missing result.



(e) Response times.

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

We can observe in Figure 4 that the results are similar to those obtained in the previous section, which demonstrates the efficiency of our algorithms to perform in DISAP as well as in systems



(a) (b)

Figure 5: Created backup queries for different workloads and: (a) two different levels of query's criticality; (b) two different providers' failure probability.

with non-autonomous participants. More interestingly, we can see in Figure 4(a) that both S_bQR and S_bQR+ create much more backup queries than in previous results, especially those having a high criticality (see Figure 4(b)). 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 Figure 4(a)). An interesting point to highlight in these results is that S_bQR+ voluntary abort more queries than it misses due to providers' failure. This proves its capacity to deal with providers' failure. Now, in Figure 4(d), we can observe 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 can observe in Figure 4(e) that, even though S_bQR and S_bQR+ create now more backup queries, their ensured response time is only degraded by 70 milliseconds in average. This proves their effectiveness for environments such as cluster-based systems.

Comparing these results with those of the previous section, we can conclude that the previous results were indeed impacted by participants' preferences. In the previous section, we can see that the number of backup queries created by S_bQR and S_bQR+ is lower in order to avoid participants' dissatisfaction, which proves the adaptation of our algorithms to participants' intentions.

6.2.2 Varying queries' criticality

To analyze the sensitivity of our algorithms to different queries' criticities, we present the results of two series of experiments: one with lowly critical and the other one with highly critical queries, i.e. with a criticality of 0 and 1 respectively. Notice that the criticality of queries does not impact $replicateAll$'s nor $none$'s results because they do not consider this criteria. This is why we only show the results for our algorithms. For these experiments, we assume again that participants have again some preferences as stated in the setup section. Figure 5(a) illustrates the number of created backup queries for different workloads. These results confirm the results of Figure 6.2.2 in the fact that our algorithms tend to replicate less queries as the workload increases in order not

to overload providers. We can also observe the sensitivity of our algorithms to queries' criticality by creating more backup queries for the highly critical queries. We confirm that, as stated in Section 5.2, S_bQR+ never replicates more queries than S_bQR. As the workload increases, the number of queries replicated by S_bQR+ is much smaller than those replicated by S_bQR. Once 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. This results demonstrate the self-adaptability of our algorithms to the queries' criticality by modifying on-line the rate at which queries are replicated.

6.2.3 Varying providers' failure probability

We finally run S_bQR and S_bQR+ 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). Moreover, we assume that queries arrive with a criticality of 1. For clarity of results, we assume that consumers ask for a single answer per query, i.e., $n = 1$. Because of this last assumption and S_bQR+'s properties (see Section 5.2), the results for S_bQR are the same as for S_bQR+, so the results we present here are valid for both of them.

In Figure 5(b), 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 providers' failure probability. We also observed in our experimentations that our algorithms have in average less queries with missing results than $none$ and $replicateAll$.

Concerning participants' satisfaction (we do not illustrate the results because of space reasons), we observed that our algorithms better satisfy participants than other algorithms in most cases. In some cases, consumers feel better satisfied with other algorithms, but the difference is quite small (similar to Figure 3(f)). Finally, we could also observe that our algorithms always better satisfy providers than other algorithms.

7 Conclusion

In this paper, we considered query replication in Distributed Information Systems with Autonomous Participants (*DISAP*). In particular, we focused on query replication based on the active replication model. Query replication is hard to support in DISAP because it may decrease system performance and may also dissatisfy providers. With autonomous participants, the only way to avoid having several participants to voluntarily leave the system is to satisfy their interests. This is why we studied query replication from a satisfaction point of view. Our analysis of query replication reveals that it is useful to address two different problems: to tolerate providers' failures and to deal with Byzantine providers. To our knowledge, this is the first work that analyzes active query replication in its whole generality and from a satisfaction point of view. In summary, our main contributions are the following.

First, we proposed a satisfaction model that considers participants' failures. In particular, we defined a notion of consumer's satisfaction which considers the fact that queries have different criticality and that a consumer may receive less results than it expects because of providers' failures. We also defined a notion of provider's satisfaction, which considers the fact that a provider may perform queries for nothing because of backup queries or consumers' failures. Then, to complete the satisfaction model, we defined a notion of global satisfaction that considers participants' satisfaction and their probability of failure.

Second, we proposed a query replication algorithm, S_bQR, which deals with participants' failures by deciding on-line whether a query should be replicated and at which rate. A salient feature of S_bQR is that it replicates only those queries that allow increasing global satisfaction. Third, we proposed another query replication algorithm, called S_bQR+, which also considers the replication required by a consumer because of Byzantine providers. Generally speaking, S_bQR+ generalizes S_bQR with the goal of prioritizing critical queries. To this end, it may voluntarily fail to allocate as many replicas as required by consumers for their low critical queries so as to keep resources for the critical ones. An important feature of both S_bQR and S_bQR+ algorithms is that they make no assumption on how many providers' failures can occur at any time.

Finally, we implemented both algorithms and compared them to the popular baseline algorithm which replicates a query to all providers (the $replicateAll$ algorithm). Our experimental results demonstrated that our algorithms significantly outperform $replicateAll$ from the performance and satisfaction points of view. In particular, we demonstrated that our algorithms dynamically adapt to the workload and ensures good system's performance even for heavy workloads. Also, the results show that while replicating systematically all queries suffers from serious performance problems, both S_bQR and S_bQR+ correctly adapt the query replication rate to queries' criticality and participants' failure probabilities in order to ensure good system's performance and high participants' satisfaction. Furthermore, we demonstrated that while $replicateAll$ loses in average medium and high-critical queries, our algorithms mainly miss results for lowly critical queries.

In summary, we can conclude that, for ensuring the number of query instances required by consumers in DISAP, S_bQR is the most adequate algorithm to replicate queries. But, for increasing participants' satisfaction as well as prioritizing highly critical queries, S_bQR+ is the best choice to select relevant providers.

References

[1] Amazon s3. http://aws.amazon.com/s3/.

[2] distributed.net project. http://www.distributed.net/.

[3] grid.org project. http://www.grid.org/.

[4] Web Services, http://www.w3.org/2002/ws/.

[5] Xgrid, http://www.apple.com/server/macosx/technology/xgrid.html.

[6] Xgrid, http://www.univaud.com/hpc/products/grid-mp/.

[7] D. P. Anderson, E. Korpela, and R. Walton. High-Performance Task Distribution for Volunteer Computing. In *Procs. of the E-SCIENCE Conf.*, pages 196–203, 2005.

[8] I. Assayad, A. Girault, and H. Kalla. A Bi-Criteria Scheduling Heuristics for Distributed Embedded Systems Under Reliability and Real-Time Constraints. In *Procs. of the DSN Conf.*, pages 347–356, 2004.

[9] M. Balazinska, H. Balakrishnan, S. R. Madden, and M. Stonebraker. Fault-Tolerance in the Borealis Distributed Stream Processing System. *ACM TODS*, 33(1):1–44, 2008.

[10] V. Berten, J. Goossens, and E. Jeannot. A Probabilistic Approach for Fault Tolerant Multiprocessor Real-Time Scheduling. In *Procs. of the IPDPS Symp.*, 2006.

[11] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding Availability. In *Procs. of the IPTPS Conf.*, pages 256–267, 2003.

[12] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1-7):107–117, 1998.

[13] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. chapter The Primary-Backup Approach, pages 199–216. ACM Press, 2nd edition, 1993.

[14] M. Castro, M. Costa, and A. Rowstron. Performance and Dependability of Structured Peer-to-Peer Overlays. In *Procs. of the DSN Conf.*, pages 9–18, 2004.

[15] B. Chandramouli, C. Bond, S. Babu, and J. Yang. Query Suspend and Resume. In *Procs. of the SIGMOD Conf.*, pages 557–568, 2007.

[16] A. Fernandez, L. Lopez, A. Santos, and C. Georgiou. Reliably Executing Tasks in the Presence of Untrusted Entities. In *Procs. of the SRDS Conf.*, pages 39–50, 2006.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung. Google File System. In *Procs. of the SIGOPS Conf.*, pages 29–43, 2003.

[18] S. Ghosh, R. Melhem, and D. Mossé. Fault-Tolerant Scheduling on a Hard Real-Time Multiprocessor System. In *Procs. of the IPDPS Conf.*, pages 775–782, 1994.

[19] A. Girault, H. Kalla, and Y. Sorel. An Active Replication Scheme that Tolerates Failures in Distributed Embedded Real-Time Systems. In *Procs. of the DSN Conf.*, pages 159–168, 2003.

[20] K. Hashimoto, T. Tsuchiya, and T. Kikuno. Effective Scheduling of Duplicated Tasks for Fault-Tolerance in Multiprocessor Systems. *IEICE Transactions on Information and Systems*, E85-D(3):525–534, 2002.

[21] J. Kim, H. Lee, and S. Lee. Replicated Process Allocation for Load Distribution in Fault-Tolerant Multicomputers. *IEEE Transactions on Computers*, 46(4):499–505, 1997.

[22] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. Seti@home: Massively Distributed Computing for SETI. *Computing in Science and Engineering*, 3(1):78–83, 2001.

[23] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[24] D. Kuokka and L. Harada. Matchmaking for Information Agents. In *Procs. of the IJCAI Conf.*, pages 672–678, 1995.

[25] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM TPLS*, 4(3):382–401, 1982.

[26] L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. *Journal of Electronic Commerce*, 8(4):39–60, 2004.

[27] L. Ni and A. Harwood. A Comparative Study on Peer-to-Peer Failure Rate Estimation. In *Procs. of the ICPADS Conf.*, pages 1–7, 2007.

[28] F. Pentaris and Y. Ioannidis. Query Optimization in Distributed Networks of Autonomous Database Systems. *ACM TODS*, 31(2):537–583, 2006.

[29] S. Pieper, J. Paul, and M. Schulte. A New Era of Performance Evaluation. *IEEE Computer*, 40(9):23–30, 2007.

[30] H. Qu, A. Labrinidis, and D. Mosse. UNIT: User-centric Transaction Management in Web-Database Systems. In *Procs. of the ICDE Conf.*, pages 1–10, 2006.

[31] J.-A. Quiané-Ruiz, P. Lamarre, and P. Valduriez. A Self-Adaptable Query Allocation Framework for Distributed Information Systems. *VLDB Journal*, Online first. DOI: 10.1007/s00778-008-0114-1.

[32] E. Rahm and R. Marek. Dynamic Multi-Resource Load Balancing in Parallel Database Systems. In *Procs. of the VLDB Conf.*, pages 395–406, 1995.

[33] M. Roussopoulos and M. Baker. Practical Load Balancing for Content Requests in Peer-to-Peer Networks. *Distributed Computing*, 18(6):421–434, 2006.

[34] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measuring and Analyzing the Characteristics of Napters and Gnutella Hosts. *Multimedia Systems*, 9(2):170–184, 2003.

[35] F. Schneider. *Distributed Systems*, chapter Replication Management Using the State-Machine Approach, pages 169–197. ACM Press, 2nd edition, 1993.

[36] S. Shatz, J.-P. Wang, and M. Goto. Task Allocation for Maximizing Reliability of Distributed Computer Systems. *IEEE Transactions on Computers*, 41(9):1156–1168, 1992.

[37] U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query Optimization over Web Services. In *Procs. of the VLDB Conf.*, pages 355–366, 2006.

[38] M. Stonebraker, P. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A Wide-Area Distributed Database System. *VLDB Journal*, 5(1):48–63, 1996.

[39] G. Wolf, H. Khatri, B. Chokshi, J. Fan, Y. Chen, and S. Kambhampati. Query Pprocessing Over Incomplete Autonomous Databases. In *Procs. of the VLDB Conf.*, pages 651–662, 2007.

[40] H. Yu and A. Vahdat. The Costs and Limits of Availability for Replicated Services. In *Procs. of the SOSP Conf.*, pages 29–42, 2001.

A Proof of Theorem 1

Proof 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:*

$$\sum_{j=1}^{r+1}\left(\mathcal{A}_q^{\overrightarrow{R}_q[j]}\cdot\mathcal{A}_q^c\cdot\mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}})\cdot\overrightarrow{PI}_q[\overrightarrow{R}_q[j]]\right)-$$
$$\sum_{j=1}^{r}\left(\mathcal{A}_q^{\overrightarrow{R}_q[j]}\cdot\mathcal{A}_q^c\cdot\mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}})\cdot\overrightarrow{PI}_q[\overrightarrow{R}_q[j]]\right)$$

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

$$\mathcal{A}_q^{\overrightarrow{R}_q[r+1]}\cdot\mathcal{A}_q^c\cdot\mathcal{S}_q^{n-1}(\widehat{P_q^r})\cdot\overrightarrow{PI}_q[\overrightarrow{R}_q[r+1]]$$

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:*

$$\sum_{j=1}^{r+1}\left(\mathcal{A}_q^{\overrightarrow{R}_q[j]}\cdot\left(1-\mathcal{A}_q^c\right)\cdot\mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}})\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[j]]\right)-$$
$$\sum_{j=1}^{r}\left(\mathcal{A}_q^{\overrightarrow{R}_q[j]}\cdot\left(1-\mathcal{A}_q^c\right)\cdot\mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}})\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[j]]\right)$$

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

$$\sum_{j=1}^{r+1}\left(\mathcal{A}_q^{\overrightarrow{R}_q[j]}\cdot\left(1-\mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}})\right)\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[j]]\right)-$$
$$\sum_{j=1}^{r}\left(\mathcal{A}_q^{\overrightarrow{R}_q[j]}\cdot\left(1-\mathcal{S}_q^{n-1}(\widehat{P_q^{j-1}})\right)\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[j]]\right)$$

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:*

$$\mathcal{A}_q^{\overrightarrow{R}_q[r+1]}\cdot\left(1-\mathcal{A}_q^c\right)\cdot\mathcal{S}_q^{n-1}(\widehat{P_q^r})\cdot\overrightarrow{PC}_q[\overrightarrow{R}_q[r+1]]$$

and

$$\mathcal{A}_q^{\overrightarrow{R}_q[r+1]} \cdot \big(1 - \mathcal{S}_q^{n-1}(\widehat{P_q^r})\big) \cdot \overrightarrow{PC}_q[\overrightarrow{R}_q[r+1]]$$

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:*

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

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:*

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

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:

$$\mathcal{A}_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \sum_{a=1}^{r+1} \Big(\mathcal{S}_q^a(\widehat{P_q^{r+1}}, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[a]] \Big) \right) - \\ \mathcal{A}_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \sum_{a=1}^{r} \Big(\mathcal{S}_q^a(\widehat{P_q^r}, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[a]] \Big) \right)$$

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

$$\mathcal{A}_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \Big(\sum_{a=1}^{r} \big(\mathcal{S}_q^a(\widehat{P_q^{r+1}}, j) - \mathcal{S}_q^a(\widehat{P_q^r}, j) \big) \cdot \overrightarrow{CI}_q[a] \Big) \right)$$

and separating the $r+1$ *value,*

$$\mathcal{A}_q^c \cdot \sum_{j=0}^{n} \Big(\Delta_q^j \cdot \frac{1}{n} \cdot \mathcal{S}_q^{r+1}(\widehat{P_q^{r+1}}, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[r+1]] \Big)$$

Thus, we have the following equation:

$$\mathcal{A}_q^c \cdot \sum_{j=0}^{n} \left(\Delta_q^j \cdot \frac{1}{n} \cdot \Big(\sum_{a=1}^{r} \big(\mathcal{S}_q^a(\widehat{P_q^{r+1}}, j) - \mathcal{S}_q^a(\widehat{P_q^r}, j) \big) \cdot \overrightarrow{CI}_q[a] + \\ \mathcal{S}_q^{r+1}(\widehat{P_q^{r+1}}, j) \cdot \overrightarrow{CI}_q[\overrightarrow{R}_q[r+1]] \Big) \right)$$

which demonstrate the fifth line (of Theorem 1).

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399