CANPRO: A Conflict-Aware Protocol for Negotiation of Cloud Resources and Services

Marco A. S. Netto

IBM Research
Sao Paulo, Brazil

Abstract. In a Cloud environment, users face the challenge of selecting and composing resources and services from a single or multiple providers. As several negotiations can occur concurrently, information on service and resource availability may be out-of-date, thus requiring several iterations between users and providers until an agreement is achieved. To address this problem, we introduce CANPRO, a Conflict-Aware Negotiation Protocol for allocating Cloud resource and services aimed at reducing cancellation messages during negotiation. CANPRO allows users (or entities on their behalf) to know the amount of resources being concurrently negotiated by other users and the number of users interested in such an amount, while still keeping users' information private. By knowing this information, users can, for instance, confirm allocation requests with lower chances of having collisions with other users. In addition, for the same reason, users can increase their time deciding which (combination of) resources they want to allocate. The paper presents comparative results of CANPRO against the popular two-phase commit protocol (2PC) and a state-of-the-art protocol named SNAP-3PC. We used think time, network overhead, number of concurrent negotiations and providers as main metrics. The results are promising and the protocol can be used in scenarios other than Cloud Computing; for instance, bookings of health services, cars, tickets for venues, schedule of appointments, among others.

1 Introduction

Users have access to several services in the Internet to perform tasks that range from exchanging e-mails to allocating high performance computing resources. Some of these services and resources are offered by Cloud providers using a pay-as-you-go model. As the number of these providers and services increases, users face the challenge of selecting, and possibly, composing them in a complex and dynamic computing environment.

Several users may request resources and services at the same (or within the same time interval). During negotiations, information about resource availability, from the moment of selecting to the moment of confirming the allocations, may be out-of-date. This has a particular impact when users are composing services from multiple providers, as a failure in a single allocation may result in a renegotiation with all the resource providers. The failure generates a phenomenon

called *livelock*, in which multiple users keep trying to allocate resources over and over gain without success, thus requiring a considerable number of negotiation messages to satisfy all users. This problem is well-known and investigated in the area of Distributed Transactions [2], mainly investigated in the data base community and in the last decade in the Grid community [10]. For Cloud Computing, such renegotiation may cause violation of Service Level Agreements (SLAs) for confirmed requests.

A number of protocols for allocating distributed resources have been proposed in the literature [5–8, 10–12], being most of them aimed at avoiding deadlocks and livelocks, and reducing the number of messages during negotiations. These protocols consider that a user is not aware that other users are concurrently negotiating for the same resources—the user only receives a message that was not possible to commit the selected resources. Therefore, users have no chance of optimizing their resource selection in order to avoid such a competition. As a consequence, negotiations require several messages and users have little time to make their allocation decisions.

The main contribution of this paper is the Conflict-Aware Negotiation Protocol (CANPRO), which aims at reducing the number of cancellation messages during negotiations and increasing the time users can spend to decide which resources they want to allocate. This is achieved by allowing users (or entities on their behalf) to know the amount of resources being concurrently negotiated by other users and the number of users interested in such an amount. In other words, all users are aware about intentions of other users concurrently negotiating for conflicting resources. By knowing this information, users can, for instance, confirm allocation requests with lower chances of having collisions with other users. In addition, for the same reason, users can increase their time deciding which (combination of) resources they want to allocate. We compare CANPRO against the popular two-phase commit (2PC) protocol and a state-of-the-art protocol named SNAP-3PC, which is a three-phase commit extension of the Service Negotiation and Acquisition Protocol. The results are promising and the protocol can be used in scenarios other than Cloud Computing.

2 Conflict-Aware Negotiation Protocol

The Conflict-Aware Negotiation Protocol (CANPRO) allows users to be aware about concurrent negotiations that conflict with one another. Whenever a provider receives a message from a user requesting for resources, this provider sends back to the user a message describing: (1) whether the requested capacity is available; (2) the percentage of that requested capacity that conflicts with the capacity being concurrently negotiated by other users; and (3) the number of those users negotiating the conflicting capacity. The users that already received offers from the provider(s) obtain an update on conflicts, which has an influence on their decision regarding where and how many resources they should commit.

Figure 1 illustrates an example of information flow between one resource provider and two users. In this example, User_2 requests for resources just after

resource provider sends $User_1$ information about resource availability. Knowing that $User_1$ is negotiating resources, the provider sends $User_2$ information about resource availability considering the possible conflicts with $User_1$, who is also notified about a possible conflict. For this particular example, $User_1$ commits the original request, whereas $User_2$ commits only part of it in order to avoid collision. Both users receive confirmations on their commit requests.

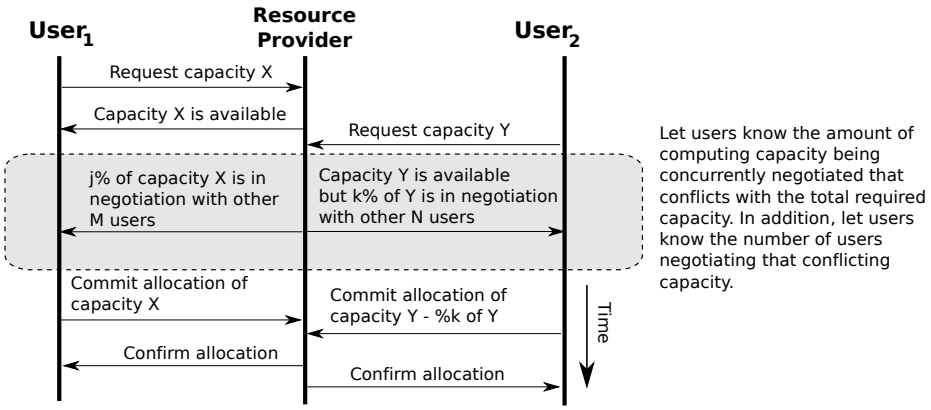


Fig. 1. CANPRO execution example.

CANPRO requires a data structure to keep track of the concurrent negotiations. Our current implementation utilizes a linked list of resource offers called *NegotiationQueue*. These resource offers [9] contain the resource availability given to a user by the provider. For CANPRO, these offers are extended to add conflict information: percentage of conflicting resources and the number of users negotiating those resources. The NegotiationQueue is updated when (i) a new request gets to the resource provider; (ii) the provider confirms the resource allocation (commit message); (iii) the provider rejects user request; (iv) the user decides not to continue the negotiation or when the user has to start a new negotiation.

Based on the NegotiationQueue, the provider can generate conflict information and notify users whenever this information changes. The conflict represents the percentage of resource being concurrent negotiated by multiple users. Before the resource provider receives the request from $User_2$, the NegotiationQueue contains only the offer given to $User_1$.

The NegotiationQueue is updated with the offer given to $User_2$ once the new request arrives. At this point, the provider triggers the algorithm to calculate conflicts and notify users if any original offer has been changed. Algorithm 1 presents a pseudo-code of how conflicts can be calculated. The algorithm receives the NegotiationQueue and returns a list of offers containing conflict information.

The variables used in the algorithm are:

Algorithm 1: Pseudo-code for generating list of offers with conflicting information based on the NegotiationQueue.

```
1  bin ← create a bin with the size of the available capacity
2  for ∀request ∈ NegotiationQueue do
3      Fill bin with requested capacity
4      if bin full = true then
5          binList.add(bin)
6          Create another bin with same capacity
7          Fill it with the remaining requested capacity
8  for ∀request ∈ NegotiationQueue do
9      conflictPortion ← numberOfConflicts ← 0
10     for ∀bin ∈ binList do
11         capacityRange ← find capacity range in bin that contains request
12         if capacityRange found = true then
13             conflictPortion ← conflictPortion + conflict part with other requests
               in this capacityRange
14             Increment numberOfConflicts with requests on this capacityRange
15     offer ← request information plus conflictPortion and numberOfConflicts
16     offerList.add(offer)
17 return offerList
```

- **bin:** data structure (e.g. array) that stores request capacity information;
- **binList:** list of bins;
- **capacityRange:** if bin is an array, capacityRange is a range index;
- **offer:** resource availability information to be sent to users;
- **offerList:** list of offers.

From the user side, once they receive offers (new or updated ones), they can select providers that sent offers with fewer number of conflicts and lower percentage of conflicting capacity.

3 Evaluation

The basis for the design of CANPRO is predicated on the idea that by users knowing that other users are negotiating the same (or a portion of their) resources, they can select a group of resources with lower chances of failures when commiting the allocation. The experimental results in this section demonstrate that the principle is sound.

We evaluated CANPRO against the popular two-phase commit protocol (2PC) and SNAP-3PC protocol. The latter protocol allows users who are negotiating for resources to be notified when the status of a resource is changed. We developed a multi-thread event-driven simulator with the implementation of the two protocols and CANPRO. The simulator receives a file containing information on user requests, such as user think time, allocation strategy (single x

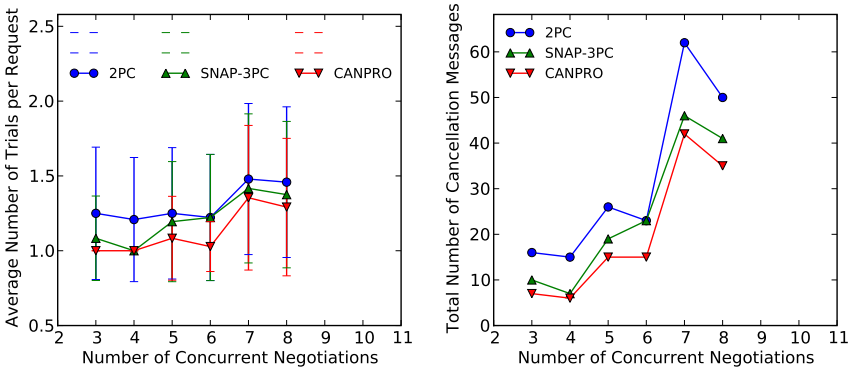
multiple providers), network delay to communicate with the providers, and the provider's processing time for the request.

We created sets of workloads that vary *network delay* to send messages, *think time*, and *processing time* that follow a Gaussian distribution. The values vary 1000±200, 2000±1000, and 1000±200 respectively (time in milliseconds). We also varied the number of concurrent requests and resource providers. For each workload, half of users requests a specific provider and the other half requests a group of providers. The total number of requests processed for the three protocols in the experiments varying all parameters is 14400. As metrics, we measured the number of trials and number of cancellation messages until a request is successfully committed.

Figures 2 (a), (c), and (e) present the number of trial allocations for three, five, and seven providers, respectively, as a function of the number of concurrent negotiations (starting with the same number of providers, and a fixed think time). The lower the number of trials the better the protocol is. The average number of trials per request increases with both the number of concurrent negotiations and the number of providers. This happens because the chances of users receiving out-of-date information on resource availability increase with these two variables. CANPRO outperforms 2PC and SNAP-3PC with the same proportion for the three numbers of providers. This indicates that the improvement scalability of CANPRO is similar to 2PC and SNAP-3PC.

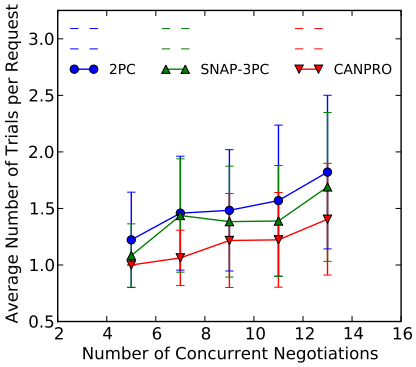
Figures 2 (b), (d), and (f) present the total number of cancellation messages for three, five, and seven providers, respectively, as a function of the number of concurrent negotiations. The behavior of this metric is similar to the previous one, however when the number of providers is three, it is observed that the higher the number of concurrent negotiations the higher the advantage of CANPRO in relation to 2PC. This happens because the number of negotiations is lower enough so that the percentage of conflicting capacity has more influence than the number of conflicting negotiations. This scenario is therefore the one where CANPRO has higher benefits.

In order to observe the effect of the think time for both negotiation protocols, we fixed the number of providers in five, and the number of concurrent negotiations as ten. As observed in Figure 3, the higher the think time the better the performance of CANPRO in relation to 2PC and SNAP-3PC. This is due to the fact that users can receive messages about negotiation conflicts while they are deciding on resource selection. For 2PC and SNAP-3PC, the higher the think time values the higher the probability of users receiving out-of-date information. As it is showed in Figure 3, 2PC and SNAP-3PC do not produce a steady performance, i.e. there is high variability when changing the average think time, whereas for CANPRO, the number of cancellations is reduced with the increase in the think time. This is a particularly promising result, as we expect users to have access to more services in the Internet, so they require time to think about their decisions.

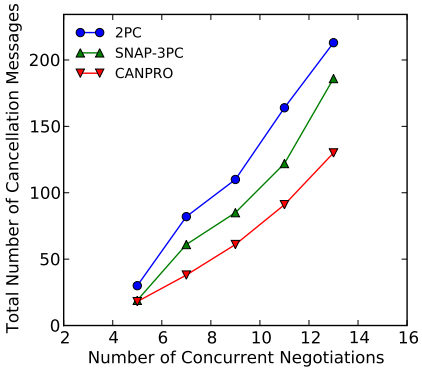


(a) Three Providers.

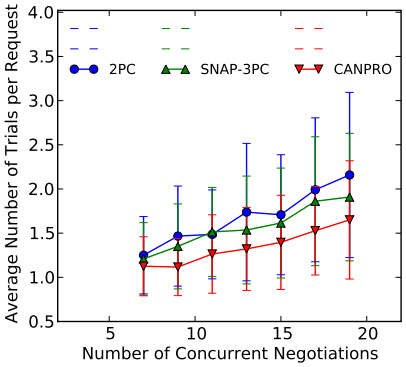
(b) Three Providers.



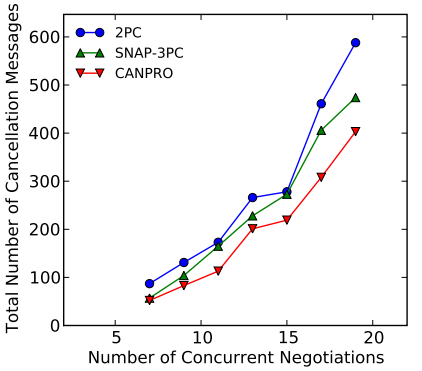
(c) Five Providers.



(d) Five Providers.



(e) Seven Providers.



(f) Seven Providers.

Fig. 2. Number of trial allocations and cancellation messages as a function of the number of providers and concurrent negotiations.

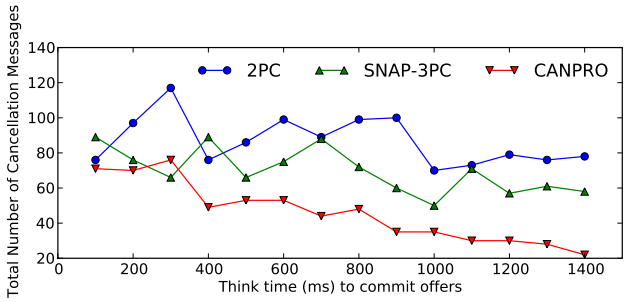


Fig. 3. Number of cancellation messages for five RPs as a function of think time.

4 Related Work

Existing protocols on resource negotiation fall into following categories: Two-Phase Commit (2PC), Three-Phase Commit (3PC), Order-based Deadline, and Polling-based protocols. Most of these projects have as motivation the problem of resource co-allocation for Grid Computing environments [10].

Kuo and Mckeown [7] presented a protocol for advance reservations and co-allocation, which extends the 2PC protocol with support for cancellations that may occur at any time. Park [11] introduced a decentralized protocol for allocating large-scale distributed resources, which is free from deadlocks and livelocks. The protocol is based on the Order-based Deadlock Prevention Protocol ODP^2, but with parallel requests in order to increase its efficiency. Another approach to avoid deadlock and livelock is the exponential back-off mechanism [6].

Takefusa et al. [12] developed a 2PC-based protocol that uses polling from the client to the server. Maclaren et al. [8] introduced a system called HARC (Highly-Available Robust Co-allocator), which uses 3PC protocol based on Paxos consensus algorithm [4] and focuses on fault tolerance aspects. Azougagh et al. [1] introduced the Availability Check Technique (ACT) to reduce the conflicts during the process of resource co-allocation. Requests wait for updates from providers until they fulfill their requirements. The main difference of ACT and CANPRO is that in the former, users are not aware about possible conflicts during negotiation, therefore it cannot optimize their resource selection decisions.

Czajkowski et al. [3] proposed the Service Negotiation and Acquisition Protocol (SNAP), which aims at managing access to and use of distributed computing resources by means of Service Level Agreements (SLAs). The protocol is not optimized to work with out-of-date information on resource availability. In order to solve this problem, Haji et al. [5] developed a 3PC protocol for SNAP-based brokers. Its key feature is the use of *probes*, which are signals sent from the providers to the candidates interested in the same resources to be aware of resource status' changes. Different from Haji et al.'s protocol, CANPRO notifies users on other concurrent negotiations, which is before the status' changes, so users can select resources with lower chances of being taken by other users.

5 Concluding Remarks

This paper presented CANPRO, a conflict-aware protocol for negotiation of Cloud resources and services. With CANPRO, users can have higher thinking time to commit requests and network communication can have higher latency. This is achieved by allowing users to be aware that there are other concurrent users negotiating for the same resources. With this conflict information in hands, users can select resources/providers with lower probability of having requests rejected. Based on experimental results, CANPRO is able to reduce cancellation messages when there are concurrent negotiations compared to 2PC and SNAP-3PC protocols. We observed that it is quite frequent to have situations where users base their resource selection decisions on out-of-date information, and a conflict-aware protocol, in this case, CANPRO, is an important tool to handle this problem. CANPRO could also be used for other scenarios such as booking of air planes, cars, health services, and scheduling of people and rooms for meetings.

References

1. Azougagh, D., Yu, J.L., Kim, J.S., Maeng, S.R.: Resource co-allocation: A complementary technique that enhances performance in grid computing environment. In: Proceedings of ICPADS (2005)
2. Bernstein, P.A., Goodman, N.: Concurrency control in distributed database systems. ACM Computing Surveys 13(2), 185–221 (1981)
3. Czajkowski, K., Foster, I.T., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Proceedings of JSSPP (2002)
4. Gray, J., Lamport, L.: Consensus on transaction commit. ACM Transactions on Database Systems 31(1), 133–160 (2006)
5. Haji, M.H., Gourlay, I., Djemame, K., Dew, P.M.: A SNAP-based community resource broker using a three-phase commit protocol: A performance study. The Computer Journal 48(3), 333–346 (2005)
6. Jardine, J., Snell, Q., Clement, M.J.: Livelock avoidance for meta-schedulers. In: Proceedings of HPDC (2001)
7. Kuo, D., Mckeown, M.: Advance reservation and co-allocation protocol for grid computing. In: Proceedings of e-Science'05 (2005)
8. Maclaren, J., Keown, M.M., Pickles, S.: Co-allocation, fault tolerance and grid computing. In: Proceedings of the UK e-Science All Hands Meeting (2006)
9. Netto, M.A.S., Buyya, R.: Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems. In: Proceedings of HCW/IPDPS (2009)
10. Netto, M.A.S., Buyya, R.: Resource co-allocation in grid computing environments. In: Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications. IGI Global (2009)
11. Park, J.: A deadlock and livelock free protocol for decentralized internet resource coallocation. IEEE Transactions on Systems, Man, and Cybernetics, Part A 34(1), 123–131 (2004)
12. Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y., Sekiguchi, S.: GridARS: an advance reservation-based grid co-allocation framework for distributed computing and network resources. In: Proceedings of JSSPP (2007)