An SMT Approach To A Multiparty Economic Scheduling Problem

Shriphani Palakodety[1], Guha Jayachandran[1], and Aditya V. Thakur[2]

[1] Onai, San Jose, CA 95129, USA {spalakod,guha}@onai.com
[2] University of California, Davis, CA 95616, USA avthakur@ucdavis.edu

Abstract

We discuss a multiparty resource allocation problem in a digital marketplace for purchase and sale of compute resources. The problem exhibits several challenging characteristics: (i) multiple agents and resources in a single matching instance, (ii) constraints that can be imposed on individual resources or a combination of resources, and (iii) short wall clock time for production settings. We model the problem as a satisfiability problem, and discuss an implementation using the Z3 SMT solver.

1 Introduction

Marketplaces allow for the efficient allocation of resources and eventual discovery of preferences [15]. With cryptocurrencies, *virtual marketplaces* can be created where participants broadcast their requests and offers for services. This transparency allows such marketplaces to be analyzed. For instance, because the information about requests and offers is universally visible, optimal matching and price computation is possible in the marketplace (for a given definition of optimality).

In this paper, we consider the Coronai cryptocurrency network [5] where computational services can be offered and requested through a virtual marketplace. For example, a participant in this network can offer a natural language processing (NLP) service that implements *named-entity-recognition* (a popular NLP application) and request *part-of-speech* (results from a Part-Of-Speech tagger, a common NLP pipeline), together comprising a *Commitment*. A commitment may be fulfilled in its entirety or not at all. Bid prices (or asks) can also be specified for the service requests and offers.

A given matching of offers and requests on this market is termed a *Schedule* here onwards. A valid *Schedule* must (i) respect the commitments (expressed using a query language discussed later) and (ii) respect the bid and maximum ask constraints imposed. A scheduler monitors the marketplace for broadcast commitments and generates a valid schedule. The scheduler must operate in real-time to ensure that the marketplace is not stalled.

This paper presents a formulation of this multiparty economic scheduling problem as a satisfiability problem. We have successfully deployed a scheduler that used this formulation; the constraints were solved using the Z3 SMT solver [19]. The scheduler has a very low wall-clock time and scales well to a very large number of participants. The tools and data for the experimental evaluation can be found at: `https://github.com/onai/SMTexperiments`.

The rest of the paper is organized as follows: Section 2 describes the constraints defining the multiparty economic scheduling problem; Section 3 outlines the encoding of these constraints in the SMT solver; Section 4 presents the experimental evaluation; Section 5 describes related work; and Section 6 concludes.

2 Multiparty Economic Constraints

This section describes the various type of constraints in the multiparty economic scheduling problem.

2.1 Resource-level Constraints

An individual resource on the marketplace is called a *ServiceCall*. A participant can either offer or request a particular *ServiceCall*. An example of a service is *named-entity-recognition*.

Each participant puts forth a *Commitment*. A *Commitment* is a combination of offers and requests of *ServiceCall*s. For example, a participant can put forth a *Commitment* that offers the service *named-entity-recognition*, denoted as *ner*, represented as: $\{ner : offer\}$.

A participant can also put forth a *Commitment* that offers *ner* and requests $part\text{-}of\text{-}speech$ (another service, henceforth denoted as *pos*), represented as: $\{ner : offer, pos : request\}$.

The marketplace also provides two quantifiers, OneOf and AllOf. A commitment specified as $\text{OneOf}(\{ner\text{-}a : request\}, \{ner\text{-}b : request\})$ indicates two distinct *ServiceCall*s, *ner-a*, and *ner-b*. The participant is requesting *exactly one of* these two requests to be satisfied. Similarly a commitment specified as $\text{AllOf}(\{ner\text{-}a : request\}, \{mm\text{-}b : request\})$ indicates that the participant wants *all of* these requests to be satisfied.

OneOf and AllOf are composable operators. For example, a commitment

$$\begin{aligned}\text{OneOf}(\,&\text{AllOf}(\{ner\text{-}a : request\}, \{ner\text{-}b : request\}),\\ &\text{AllOf}(\{ner\text{-}c : request\}, \{ner\text{-}d : request\}))\end{aligned}$$

indicates that *exactly one of* the two AllOfs be satisfied.

It is trivial to show that all combinations of OneOf and AllOf can be re-written as $\text{OneOf}(\text{AllOf}(service\text{-}call\text{-}a, service\text{-}call\text{-}b), \ldots, \text{AllOf}(\ldots))$; i.e., a OneOf of a collection of AllOfs and each AllOf contains individual *ServiceCall* requests or offers. The proof for this follows from boolean arithmetic. Here onwards, we will only present *Commitment*s in the format $\text{OneOf}(\text{AllOf}(\ldots), \text{AllOf}(\ldots), \text{AllOf}(\ldots))$; i.e., a *Commitment* is one of several AllOfs and each AllOf contains individual *ServiceCall* requests and offers.

2.2 Price Constraints

A schedule must also decide on a price for each *ServiceCall*. These prices are decided based on participant-specified bid and ask prices. Participants can specify a ceiling or a floor *on each* AllOf. A ceiling specifies that the cumulative payment made for requests and cumulative income from offers is below that value. The floor is analogous to the ceiling.

Consider the following AllOf constraint:

$$\{\text{AllOf}(\{service\text{-}call\text{-}a : offer\}, \{service\text{-}call\text{-}b : request\}), ceiling : 20\}$$

If this particular AllOf is scheduled, the associated ceiling requires that $price(service\text{-}call\text{-}id\text{-}b) - price(service\text{-}call\text{-}id\text{-}a) \leq 20$, where $price(x)$ is the chosen price of *ServiceCall* x. The price for each *ServiceCall* is picked by the scheduler and is part of the *Schedule*.

2.3 Uniqueness Constraints

The one additional constraint is that in a *schedule*, *only one participant* should be allowed to *offer* a particular *ServiceCall*. This constraint avoids duplicated work on the marketplace.

2.4 Schedule

The commitments for all participants in the marketplace form a set of constraints. A *schedule* is a collection of commitments such that exactly one of the AllOfs in each commitment's OneOf honors all the aforementioned constraints. Thus, the scheduler picks a set of *Commitment*s to schedule and chooses one AllOf per commitment and decides on a price for all the *ServiceCall*s to honor the pricing constraints on the chosen AllOfs.
An example of a valid schedule is:

$Commitment\ 0:$	$\{\text{AllOf}(sc\text{-}a : request, sc\text{-}b : offer,), ceiling : 20\}$
$Commitment\ 1:$	$\{\text{AllOf}(sc\text{-}a : offer,), floor : 10\}$
$Commitment\ 2:$	$\{\text{AllOf}(sc\text{-}b : request,), ceiling : 10\}$
$price(sc\text{-}a):$	10
$price(sc\text{-}b):$	10

3 SMT-based Scheduler

In this section, we describe how the multiparty economic constraints are encoded and solved by an SMT solver, such as Z3. We use the following notation:

- $\mathcal{C}$ is the set of *Commitment*s, and $\mathcal{A}_i$ is the set of AllOfs in *Commitment* i.
- $\mathcal{R}$ is the set of *ServiceCall*s that are requests.
- $\mathcal{O}$ is the set of *ServiceCall*s that are offers.
- $Cost(A_{i,j})$ is the cost ceiling or floor (based on the sign) of AllOf j in *Commitment* i.
- $Price(k)$ is the price of the *ServiceCall* k.
- $\mathcal{S}_i$ is the set of *ServiceCall*s in AllOf i.
- $\mathcal{R}_k$ and $\mathcal{O}_k$ are the sets of requests and offers for *ServiceCall* k, respectively.
- C_i is a boolean variable that indicates if a particular commitment is being scheduled.
- $A_{i,j}$ is a boolean variable that indicates if AllOf j in *Commitment* i is scheduled.
- $S_{i,j}(k)$ is a boolean variable that indicates if *ServiceCall* with id k in *Commitment* i and AllOf j is scheduled.

Using the above notation, the set of *Commitment*s can be expressed using SMT constraints as follows:

- For each commitment, exactly one of the AllOfs must be scheduled. This constraint is expressed using a pseudo-boolean clause. In Z3 parlance, a pseudo-boolean allows regular arithmetic over boolean values (treating a true value as 1 and a false value as 0): for each $i \in \mathcal{C}$, $C_i \Rightarrow \sum_j A_{i,j} = 1$.
- If a *Commitment* is not scheduled, then its constituent AllOf should not be scheduled: for each $i \in \mathcal{C}$, $\neg C_i \Rightarrow \sum_j A_{i,j} = 0$.

- Scheduling an AllOf implies the constituent *ServiceCall*s need to be scheduled as well: for each $i \in \mathcal{C}, j \in \mathcal{A}_i$, $A_{i,j} \Rightarrow \bigwedge_k S_{i,j}(k)$.
- If an AllOf is not scheduled, the constituent *ServiceCall*s should not be scheduled as well: for each $i \in \mathcal{C}, j \in \mathcal{A}_i$, $\neg A_{i,j} \Rightarrow \sum_k S_{i,j}(k) = 0$.
- If a request has been scheduled, exactly one offer must be scheduled for it: for each k, $\mathcal{R}(k) \Rightarrow \sum_k \mathcal{O}(k) = 1$.
- If a service call is being offered, then at least one request must exist for it: for each k, $\mathcal{O}(k) \Rightarrow \sum_k \mathcal{R}(k) \geq 1$.
- The schedule must be non-trivial and at least one *Commitment* must be scheduled: for each $i \in \mathcal{C}$, $\sum_i C_i \geq 1$.
- The integer-valued price variables for each *ServiceCall* must satisfy the pricing constraints on each AllOf: $(\sum_{k \in \mathcal{S}_i} \textit{is-offer}(k, \mathcal{S}_i) \cdot \textit{price}(k)) \leq \textit{ceiling}(k)$, where $\textit{is-offer}(k, \mathcal{S}_i)$ is -1 if AllOf i is a request for *ServiceCall* k, and 1 if AllOf i is an offer for *ServiceCall* k.
- We use the optimization facility of Z3 [11] to optimize the total number of commitments scheduled: $\max \sum_i C_i$. Note that this commitment makes the third constraint (at least one commitment must be scheduled) redundant. However, we keep both constraints since the optimization objective can vary based on needs.

4 Experimental Evaluation

In this section, we present an empirical evaluation of the SMT-based scheduler and compare it to a prior baseline approach; the data to reproduce this evaluation can be found at: `https://github.com/onai/SMTexperiments`. In our model, the following values govern the number of variables in the resulting SMT problem:

- the number of *ServiceCall*s in a single AllOf,
- the number of AllOfs in each *Commitment*,
- the overall number of *Commitment*s to be scheduled, and
- the total number of *ServiceCall*s overall that need to be priced.

We present results using a synthetic dataset that vary each of these counts to produce different configurations of the problem. Specifically, we sample these values from possible maximum counts and synthesize 10 problems per configuration. We use the following definitions:

- C_{AO}: the maximum number of *ServiceCall*s in a single AllOf.
- C_C: the maximum number of AllOfs in a single *Commitment*.
- N_C: the total number of *Commitment*s to be scheduled.
- N_{SC}: the total number of *ServiceCall*s that need to be priced.

C_{AO}	C_C	N_C	N_{SC}	Time (secs.)
10	10	10	10	0.03
50	10	10	10	0.08
100	10	10	10	0.20
100	10	100	10	2.02
100	10	1000	10	26.76
50	10	100	100	1.29
100	10	100	100	1.98
100	10	1000	100	22.18
50	100	100	1000	1.27
100	100	100	100	22.41
100	100	1000	1000	257.60

Table 1: Performance of the SMT-based scheduler. The leftmost four columns shows the settings for the various parameters, and the rightmost column shows the wall-clock time for execution. The total number of variables in the shown configurations ranges from a few dozen to a few million.

A synthetic dataset is generated using the following procedure:

```
input  : $C_{AO}$,$C_C$,$N_C$,$N_{SC}$
output: A set of random $Commitments$
for $c \leftarrow 0$ to $N_C$ do
    $n_{allof} \sim \texttt{Uniform}(0, C_C)$
    for $n_a \leftarrow 0$ to $n_{allof}$ do
        $n_{scall} \sim \texttt{Uniform}(0, C_{AO})$
        for $n_s \leftarrow 0$ to $n_{scall}$ do
            $I(request) \sim \texttt{Bernoulli}(p)$
            $s \sim \texttt{Uniform}(0, N_{SC})$
```

Each dataset contains a total of N_C *Commitments*, each of which contains a maximum of C_{AO} AllOfs. Each of those AllOfs contains a maximum of C_{AO} *ServiceCalls*. Each of these *ServiceCalls* is picked from an existing set of *ServiceCalls* of size N_{SC}. In addition, we sample the ceilings and floors on the prices from a maximum of 200. The value of p (the parameter for the Bernoulli distribution is set to 0.5).

Platform All experiments were run on a Macbook Pro with an Intel(R) Core(TM) i7-5557U CPU @ 3.10GHz with 4 cores. The Z3 model was implemented in the Rust programming language [7] using existing libraries to interface with Z3[9].

Table 1 reports the runtime of the SMT-based scheduler for a variety of configurations, and shows that the implementation is performant across a variety of settings. The total number of variables in the shown configurations ranges from a few dozen to a few million.

4.1 Baseline

We now present results using an alternate approach, which was the initial approach used before switching to the SMT-based scheduler described above. For this baseline, we drop the pricing constraints. The pricing constraints were solved using an LP solver with fixed semantics.

C_{AO}	C_C	N_C	N_{SC}	Time (secs.)
10	10	10	10	5.04
50	10	10	10	4.93
100	10	10	10	6.51
100	10	100	10	4.83

Table 2: Performance of the hardware-accelerated baseline. The leftmost four columns contain configuration values, and the rightmost contains the wallclock time. This baseline encoding takes substantially longer to get to a result while solving only a subset of the overall scheduling problem.

The baseline approach represents each AllOf as a bit-vector of size $2 \cdot N_{SC} + N_C$. The entries in the bit vector are set using the following scheme:

- If the current AllOf contains *ServiceCall* i (i being the index of the call) and is offering this *ServiceCall*, then the bit indexed by $2 \cdot i$ is set to 1.
- If the current AllOf contains *ServiceCall* i (i being the index of the call) and is requesting this *ServiceCall*, then the bit indexed by $2 \cdot i + 1$ is set to 1.
- If the current AllOf is part of *Commitment* c, (c being the index of the commitment) then the bit indexed by $(2 * N_{SC}) + c$ is set to 1.

This encoding captures all the *ServiceCall*s that are being offered and requested within an AllOf, and the *Commitment* index c captures which OneOf (and thus *Commitment*) this AllOf is a part of. The encoding captures all information necessary to describe an AllOf. The scheduling task then is to pick which AllOfs need to be scheduled.

A valid schedule in this setting has the same shape as the bit-vector for an individual AllOf. The algorithm to find a schedule starts by initializing a bit-vector of appropriate length to zero, which represents the trivial schedule in which nothing is scheduled. For each AllOf to be scheduled, a new potential schedule is created by adding the relevant AllOf's bit vector to the initial schedule. Thus at this stage, there are as many potential schedules as there are AllOfs (say N_{AllOf}). Each of these potential schedules are then further updated by adding yet another AllOf one at a time (resulting in ${N^2}_{AllOf}$) potential schedules, and so on. Thus, the addition operation is a potential schedule update where another AllOf is possibly scheduled.

In the encoding for a potential schedule, the counts in the vector (which is no longer a bit-vector but an integer-valued vector) reflect the number of requests and number of offers (depending on their position), and the number of times a *Commitment* is scheduled. Thus, we can trivially eliminate invalid some potential schedules. Ones that offer a *ServiceCall* more than once, for example, are eliminated by the procedure `FilterDoubleOffer`, which is run each time the set of potential schedules is updated.

At each stage, a pruning step cuts down the total number of potential schedules to a reasonable number (using an appropriate heuristic such as random selection). At the end of the iterations, a final procedure deletes all invalid schedules (using the rules mentioned in Section 3). The set of hypothetical schedules maintained at every stage is denoted as $\mathcal{P}$. This baseline algorithm is stated as:

```
input  : $\mathcal{A}$: set of AllOfs in the correct encoding
output: $\mathcal{P}$: final set of hypothetical $Schedule$s
$\mathcal{P} \leftarrow \{\overrightarrow{0}\}$
for $i \leftarrow 0$ to $N_{AllOf}$ do
    for $p \in \mathcal{P}$ do
        for $a \in \mathcal{A}$ do $\mathcal{P} \leftarrow \mathcal{P} \cup \{p + a\}$
    FilterDoubleOffer($\mathcal{P}$)
FilterInvalid($\mathcal{P}$)
```

The final schedule is retrieved by keeping track of the path used to arrive at the resulting schedule; i.e., which AllOf was added at what stage. This is achieved by simple backtracking.

Platform The experiments for the baseline were run on a NVidia Tesla K40c graphic card. The baseline was implemented in Rust using the arrayfire library [1].

Table 2 presents results for a hardware accelerated implementation (targeting GPUs) of the above baseline algorithm with backtracking. Due to the nature of hardware accelerated code, a solution must pre-allocate memory for the encoding. We set the value of N_{SC} to 10, N_{AO} to 100, N_C to 100, and the size of $\mathcal{P}$ to 100 (the biggest problem size possible). As we can see in Table 2, the baseline encoding is not efficient and the Z3 solution can solve substantially bigger problems in size. We report results for only a subset of problems (that can fit in the chosen sizes). The implementation also suffers from having to copy a large amount of data (all of $\mathcal{P}$) to and from the GPU after each stage. In addition, this encoding does not include any pricing information and this would need to be computed later on for a given *Schedule* (and possibly a new *Schedule* needs to be discovered).

4.2 Varying Demand And Supply

Typical marketplaces include varying demand and supply levels. We test two additional configurations - abundance and scarcity. In the case of abundance, there is a surplus of offers compared to requests, and similarly in a scarcity setting there are not enough offers to satisfy requests. We synthesize two additional test benches where (i) the number of requests on offer are only 25% of the number of offers (the abundance setting), and (ii) the number of offers are 25% of the requests in the system. This is achieved by setting the value of p in the sampling algorithm above to 0.25 and 0.75, respectively.

Table 3 shows the results of this experiment on a subset of the original configurations. We note that the abundance setting seems to take significantly longer than the scarcity setting. One possible reason can be that in a scarcity setting, the pool of possible solutions is smaller (and thus less work for the solver).

5 Related Work

A traditional setting for job scheduling is computer clusters. Jobs or tasks are scheduled onto available cores by a scheduler. Popular grid scheduling tools are the Univa grid engine [8], Open grid scheduler [6], and HTCondor [3]. The request-offer model in these systems in not as rich as the marketplace discussed in this paper. The resources are supplied by the manager of the grid and the scheduling algorithms are some variant of a greedy scheduler. Beyond simple constraints like duration and priority (both of which can be easily implemented in a

C_{AO}	C_C	N_C	N_{SC}	Abundance Setting Time (secs.)	Scarcity Setting Time (secs.)
10	10	10	10	0.1	0.18
50	10	10	10	0.51	0.51
100	10	10	10	0.83	0.4448
100	10	100	10	12.47	5.904
100	10	1000	10	87.2815	47.0092

Table 3: Performance of the SMT-based scheduler in abundance and scarcity settings. The leftmost four columns show the settings for the various parameters, the fifth and sixth columns show the wall-clock time for execution.

greedy setting), these schedulers work with a less complex model of tasks and do not have any constraints like pricing and quantifiers to deal with. A grid network like the systems above is also a subset of the marketplace discussed in this paper (since a grid service can be a computational service offered in the marketplace). Job-schedulers are also a core component of modern container orchestration systems such as Kubernetes [4] and Docker swarm [2]. Both these systems use a greedy scheduling strategy. SMT solvers, such as Z3 [19], have been used to solve network configuration problems [18, 14, 17], and scheduling problems [10, 12, 13, 16].

6 Conclusion

This paper describes an SMT-based solution to a multiparty economic scheduling problem occurring in a digital marketplace such as with the Coronai network [5]. We demonstrate the effectiveness of our SMT-based scheduler using a variety synthetic benchmarks, and contrast it with a baseline that uses the hardware-accelerated arrayfire library to solve a relaxed version of the problem.

References

[1] Arrayfire: Overview. http://arrayfire.org/docs/, accessed: 2019-02-03

[2] Docker swarm. https://docs.docker.com/engine/swarm/, accessed: 2019-02-03

[3] Htcondor - high throughput computing. https://research.cs.wisc.edu/htcondor/, accessed: 2019-02-03

[4] Kubernetes - production-grade container orchestration. https://kubernetes.io/, accessed: 2019-02-03

[5] Onai. https://www.onai.com, accessed: 2019-06-19

[6] Open grid scheduler. http://gridscheduler.sourceforge.net/, accessed: 2019-02-03

[7] Rust programming language. https://www.rust-lang.org/, accessed: 2019-02-03

[8] Univa grid engine. http://www.univa.com/products/, accessed: 2019-02-03

[9] z3-rs: High-level rust bindings to the z3 smt solver. https://github.com/graydon/z3-rs, accessed: 2019-02-03

[10] Amari, I., Rebaya, A., Gasmi, K., Hasnaoui, S.: An optimal scheduling algorithm for data parallel hardware architectures. In: Internet of Things, Embedded Systems and Communications (IINTEC), 2017 International Conference on. pp. 111–117. IEEE (2017)

[11] Björner, N., Phan, A.D., Fleckenstein, L.: νz-an optimizing smt solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2015)

[12] Cheng, Z., Zhang, H., Tan, Y., Lim, Y.: A framework for scheduling real-time systems. In: The 22nd International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). CSREA Press (2016)

[13] Craciunas, S.S., Oliver, R.S.: Combined task-and network-level scheduling for distributed time-triggered systems. Real-Time Systems **52**(2), 161–200 (2016)

[14] Duan, Q., Al-Haj, S., Al-Shaer, E.: Provable configuration planning for wireless sensor networks. In: Proceedings of the 8th International Conference on Network and Service Management. pp. 316–321. CNSM '12, International Federation for Information Processing, Laxenburg, Austria, Austria (2013), http://dl.acm.org/citation.cfm?id=2499406.2499457

[15] Häubl, G., Murray, K.B.: Preference construction and persistence in digital marketplaces: The role of electronic recommendation agents. Journal of Consumer Psychology **13**(1-2), 75–91 (2003)

[16] Kasi, B.K., Sarma, A.: Cassandra: Proactive conflict minimization through optimized task scheduling. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 732–741. ICSE '13, IEEE Press, Piscataway, NJ, USA (2013), http://dl.acm.org/citation.cfm?id=2486788.2486884

[17] Kong, W., Li, M., Han, L., Fukuda, A.: An smt-based accurate algorithm for the k-coverage problem in sensor network. In: UBICOMM 2014 - 8th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies. pp. 240–245. International Academy, Research and Industry Association, IARIA (2014)

[18] Kovásznai, G., Biró, C., Erdélyi, B.: Generating optimal scheduling for wireless sensor networks by using optimization modulo theories solvers. In: Proc. Int. Workshop on Satisfiability Modulo Theories (SMT) (2017)

[19] de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2008)