Cross-Monotonic Game for Self-organized Context-Aware Placement of Services with Information Producers and Consumers

Manuel Osdoba(✉) and Andreas Mitschele-Thiel

Integrated Communication Systems Group,
Technische Universität Ilmenau, Ilmenau, Germany
{manuel.osdoba,andreas.mitschele-thiel}@tu-ilmenau.de
http://www.tu-ilmenau.de/iks

Abstract. Deploying service instances in a network requires multiple considerations. Firstly, an instance should be placed near clients that need the service. Secondly, it should be in the centre of those clients. Thirdly, the service provider himself should benefit from placing additional service instances.

We approximate problem one and two by a distributed auction. The winners of the auction are agents that bid to join in a cost-sharing scheme with cross-monotonic cost shares to solve problem three. Service instances that have an appropriate number of clients that consume may serve a context. Those with clients that are passive or mainly produce information may not serve the context because they can not pay their cost shares and thus would not be beneficial to the service provider. Clients of those instances (producers) are directly connected to the service providers central server. Our algorithm fits well to services with a regular consumer/producer ratio of 0.75/0.25.

Keywords: Service placement · Self-organization · Sparse knowledge · Multicast game · Facility location problem · K-mean/median-problem

1 Introduction

Resources in mobile networks are limited and the concurrent transfer of duplicate data is a waste of resources. If the flood of information is big enough, every network will suffer from the concurrent transfer of duplicate data. Therefore, it is of use to replicate or cache data at distinct points in the network. However, replicas introduce additional synchronization traffic to keep the different copies consistent. As long as the content is not changed, no synchronization mechanism is triggered. We call nodes that *read* data *consumers*. On the other hand, there are nodes that *alter/write* data. Those are *producers*. Updated information is assumed to be propagated through the entire network until all active instances that store the information received it.

Nowadays algorithms replace static mechanisms and human intervention in networks. A cloud infrastructure is usually controlled by central administrative

J. Cheng et al. (Eds.): GameNets 2016, LNICST 174, pp. 34–42, 2017.
DOI: 10.1007/978-3-319-47509-7_4

entities that orchestrate an entire network. However, clouds as well as service- and content delivery are also of relevance in wireless networks and in those networks, entities have sparse knowledge on the topology. That was our motivation do develop a self-organized algorithm that deals with the challenges that were stated in the abstract.

The paper is organized as follows. In Sect. 2, we study previous approaches that influenced our mechanism. In the following part, our model and assumptions as as well as definitions on cost sharing games and requirements are given. Section 3 deals with our cross-monotonic algorithm, which is evaluated in Sect. 5. We conclude the results, shortcomings and advantages in Sect. 6.

2 Related Work

Our approach addresses two challenges in future networks. Service instances need to be placed near clients that request the service. Usually, a service provider analyzes the clients behavior and the decision where to place service instances is based upon an offline optimization. There were several approaches that addressed an online optimization of wireless (sensor) networks, like [11].

Games for Service Placement. In wireless networks, environmental factors need to be monitored. For that purpose, Wu et al. [11] created a submodular game that improves Quality of Monitoring. Once a wireless sensor node notices a lack of monitoring, it solves a knapsack problem to find out, whether it has enough resources to run an additional monitoring application. An appropriate per-node utility function considers a neighbors allocation and maximizes the social welfare that is measured by the quality of monitoring. Whenever a node changes its strategy, it has to send its modified strategy to its neighbors.

Different kinds of facility location and k-Median problems were addressed by Pál et al. in [8]. In their Single-Source Rent-or-Buy game, a strategy-proof cost-sharing scheme was proposed. The edges of a Steiner tree are bought while edges on the shortest path from a receiver to the Steiner tree are rented. Every individual packet transfer along a rented edge has to be paid separately. The costs that arise if an edge is bought, are *shared* among the receivers in the Steiner component. Furthermore, their ghost-mechanism requires the receivers to continuously fund a fraction of their cost shares to establish the Steiner tree.

The second challenge is to describe and identify the context of nodes.

Context-Awareness. Self-Optimization and Self-Configuration is an essential functionality in the Internet of Things. Several *things* can share a context. In [9] is described how the correlation among contexts can be measured. Jaffal et al. [3] analyzed, in which way a context can be abstractly described to aid in the design of pervasive systems. Najar et al. [5] elaborated on how a service, that satisfies a clients intention in a given context, can be chosen. Especially [1,6] analyze an architecture that includes contextual awareness as a factor in deploying and designing services.

3 System Model and Requirements

Data for *consumers* can be retrieved from the node that stores all data or at lower cost (and lower delay) from a nearby service instance. If the data item was previously retrieved, *consumers* get their information from a nearby service instance and the retrieval from the central node can be omitted. A high cache hit rate results in a decreased use of the connection to the central node. It increases if it is likely that clients request same data and that is the case if those are correlated. We see two ways to find out whether data is correlated.

Firstly, data can be partitioned. Hence, requests are correlated if clients request data from the same partition. Secondly, for unpartitioned/unstructured data, we use methods of big data and data analysis. The information, a client is interested in can be put into a selector vector. It has to be staged by the service/content provider (e.g. by formal concept analysis [3]) and is used by the facilities to discriminate clients within the same context from those who are not. The comparison of a selector vector with a client vector can be done with the symmetric Kullback-Leibler- or symmetric Jensen-Shannon divergence [1–3,5,6,9]. Matching a clients behavior to the selector vector may require deep packet inspection on samples of the traffic. Furthermore, a candidate needs to be aware of the clients profile in the service. These can be *information consumers* or *information providers*. The occurring system costs are shown in Eq. 1

$$\Omega(S \subseteq F, t) = \sum_{v \in S} \sum_{c \in C_v} (\boldsymbol{r}_c(t) \cdot (\text{cost}(\text{P}_{c,s}) - \text{cost}(\text{P}_{c,v})) + \boldsymbol{w}_c(t)\mathcal{C}_{Steiner}) \quad (1)$$

In Fig. 1, all clients have to retrieve their data from the central instance s. If node u in Fig. 1 *reads* data, it incurs cost of $\sum_{e \in P_{us}} \text{cost}(e) = 18$. If he acts as an producer and *writes* to s, costs will be 18, equally. If the same scenario occurs in Fig. 2, node u updates data that is distributed among the nodes that are colored green, it incurs cost of 43. If another user requests the updated content, the data is retrieved from the local instance. If $\lceil (40 - 15)/20 \rceil = 2$ users in the

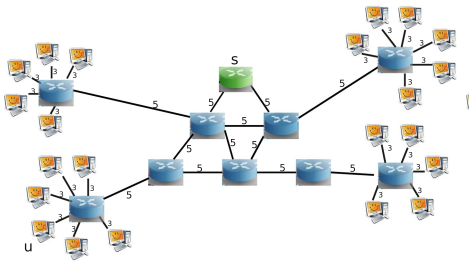

Fig. 1. Only one service instance is active (=green): Configuration if majority of operations is *Write* (Color figure online)

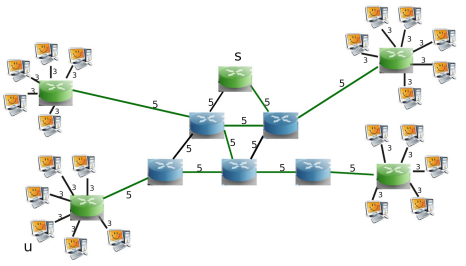

Fig. 2. Several service instances are activated (=green) and connected by Multicast Tree: configuration if majority of operations is *Read* (Color figure online)

lower right corner of the picture read the new content, costs are compensated. Our approach suits networks in which nodes have sparse knowledge. Therefore, centralized algorithms are inappropriate. To find nodes that are appropriate to serve clients, the k-Median problem is approximated with best response dynamics. If consumers and producers change their behavior, service instances adapt accordingly. Consumers at a service instance compensate the costs that arise on the reception of the updates. If the amount of consumers is too low, those costs are not compensated and the instance stops serving that context. We optimize both with a cost sharing scheme.

Definition 1 (Cost-Sharing Scheme [7]). *A cost-sharing scheme is a function* $\xi : \mathcal{A} \times 2^{\mathcal{A}} \rightarrow \mathbb{R}^+ \cup \{0\}$ *such that, for every* $S \subset \mathcal{A}$ *and every* $i \notin S, \xi(i, S) = 0$.

The value $\xi(i, S)$ determines the cost shares of the agent i within the set S. Usually an agent has an incentive to cover the costs it invests in cost shares. An agents revenue/gain has to be greater than its investment in the cost shares. An agent that can not cover its cost shares is pruned from the scheme, which is one important mechanism to achieve the following property of our game.

Definition 2 (Cross-Monotonicity [7]). *A cost-sharing scheme* ξ *is cross-monotonic if for all* $S, T \subseteq \mathcal{A}$ *and* $i \in S, \xi(i, S) \geq \xi(i, S \cup T)$.

Cross-monotonicity is also known under the term population monotonic [10]. The set of participating agents increases while the cost shares decrease. An agents cost share *may not rise* even if additional agents join in. An equivalent argument for cross-monotonicity is $\xi(i, S) \geq \xi(i, S')$ for all $S \subseteq S'$. Therefore cross-monotonicity stimulates other agents to join. Cost sharing schemes should provide further import characteristics like *competitiveness* and *cost recovery*.

Definition 3 (Competitiveness [8]). $\sum_{i \in S} \xi(S, i) \leq c^*(S)$ *and assures that the participating agents are not charged more than the true cost* $c^*(S)$.

If competitiveness is not assured, there would be the possibility, that some other agent could offer the service at lower cost. The following term is also known as weak budget-balance.

Definition 4 (Cost Recovery [8]). $\sum_{i \in S} \xi(i, S) \geq c^*(S)$ *and assures that the costs are recovered.*

Our game is cross-monotonic and recovers the cost of the update distribution.

4 Cross-Monotonic Semi-cost Recovering Game

A game is a triple of $(\mathcal{A}, \mathcal{S}, u_v)$ with agents $\mathcal{A}$, a strategyspace $\mathcal{S}$ and a revenue function u_v for node v. As mentioned in the previous section, the agents are

the available facilities that can host an instance of the service. The strategy, an agent may choose from, is defined as follows:

$$\mathcal{S} = \left\{2^C \times \mathbb{R} \times \mathbb{B}\right\}$$

An agent vs strategy for k correlated sets is $\mathfrak{s} = (C' \subseteq C_v, \mathfrak{b}, \mathfrak{a})^k \in \mathcal{S}^k$. In $\mathfrak{s}$, C' represents the correlated clients of a context that are near node v, $\mathfrak{b}$ is the nodes bid to serve the client set C' and $\mathfrak{a}$ is true if the clients *read*/*write* ratio fulfills Eq. 2 and therefore, the instance is efficient in decreasing the value of Eq. 1. In Eq. 2, value $\alpha_v(S, i, t)$ denotes the agents cost-shares that have to be paid at every reception of an update in context i and is depicted in Eq. 4. Agent vs strategy for k correlated sets is $\mathfrak{s} = \left((C'_0 \subseteq C_v, \mathfrak{b}_0, \mathfrak{a}_0), \ldots, (C'_{k-1} \subseteq C_v, \mathfrak{b}_{k-1}, \mathfrak{a}_{k-1})\right) \in S^k$. The available contexts inside the network are denoted by $\mathcal{K}$.

$$\sum_{c \in C'} \int_t^{t+\tau} \boldsymbol{r}_c(t) \cdot (\text{cost}(\text{P}_{c,s}) - \text{cost}(\text{P}_{c,v}) dt > \sum_{c \in C' \setminus C_v} \int_t^{t+\tau} \boldsymbol{w}_c(t) \cdot \alpha_v(S) dt \tag{2}$$

Algorithm 1 runs on every agent $v \in \mathcal{A}$. Per correlated set of clients (line 2), a node determines its median distance to the clients (line 3). The nodes bid value $\mathfrak{b}$ to serve the clients is computed with a Gaussian. We choose $\mu = 0$ and $\sigma = |C_v|$ (line 3). The argument of the Gaussian is calculated as depicted in line 4. Several approaches to the Facility Location Problem (e.g. [8]) construct a ball around the facilities or clients. At the intersection point of several balls, a facility is opened. In this approach, the facilities are agents.

A node estimates its maximum cost shares within a context with Eq. 3. $\mathfrak{C} : \mathcal{K} \rightarrow C$ represents the clients within the context.

$$\mathfrak{B}_v(i, t) = \sum_{c \in (\mathfrak{C}(i) \cap C_v)} r_c(t) \cdot \text{cost}(\text{P}_{c,s} - \text{P}_{c,v}) \tag{3}$$

Node v sends its bid $\mathfrak{B}_v(i, t)$ to serve context i to the central instance. At this point, we use scheme [7, Sect. 14.2.2, p. 367]. The central service instance decides whether the offered bid covers the cost to integrate node v into the Multicast tree. It is handled that way because we expect the central instance to know about the current *read*/*write* ratio in context i. Node vs cost shares that were determined by the referenced scheme are depicted in Eq. 4. Here, $\text{ST}(i), i \in \mathcal{K}$ is the current multicast tree of the service instances that serve context i.

$$\alpha_v(S, i, t) = \min_{w \in ST(i)} \left\{\text{cost}(P_{v,s}), \text{cost}(P_{v,w}) + \text{cost}^{ST(i)}\left(P_{w,s}^{ST(i)}\right)\right\} \tag{4}$$

Every service instance is charged the cost shares $\alpha_v(S, i, t)$ (Eq. 4) per received update. The path of node w to s in the Multicast tree is denoted as $P_{w,s}^{ST}$ and the proportionate cost of node w as cost^{ST}. The cost shares $\alpha_v(S, i, t)$ are cross-monotonic (Definition 2) if the triangle inequality holds. Furthermore, the mechanism in line 10 ascertains that the recovery of the cost of propagating the updates (Definition 4) can not be violated longer than for time τ.

```
1  if Received strategy 𝔰′ or topology change or consumer/producer change then
2      for C′ ∈ correlated_sets(v) do
3          𝔪 ← median_{c∈C′} d(c, v); σ ← |C′|;
4          δ ← Σ_{c∈C′} (𝔪 − d(c, v));
5          𝔟 ← √|C′| · e^{−δ²/(2σ²)} − ρ + 𝔯;
6          𝔰 ← (C′, 𝔟, False);
7          if 𝔟 > 0 or 𝔟 > 𝔟_{𝔰′} then
8              if strategy 𝔰 changes then
9                  Send 𝔰 to neighbors; sleep(θ);
10                 if Eq. 2 holds then
11                     Connect all c ∈ C′ to v
12                 else
13                     ∅-Strategy
14                 end
15             end
16         else
17             ∅-Strategy
18         end
19     end
20 end
```

Algorithm 1. Best response mechanism for node v

5 Evaluation

For evaluation, we investigated different *consumer/producer* profiles.

Simulation Setup. The simulations were implemented in Python with SciPy, NumPy and SimPy. For the computation of the Steiner tree among the active facilities, we used the submodular function optimizer library [4]. User operations follow a Poisson process. Each of the 80 *clients* executes a *read* or *write* operation with distinct probabilities. Exact pairs of *read/write* ratios are shown on the x-axis in Fig. 3. All users are homogeneous, meaning all users have a common *read/write*-ratio and Poisson arrival rate of 2. A node may change an item if it has previously read it. The performance is stable regarding the number of requested items or number of operations per time slot.

Results. Figure 3 shows that in the range of *read/write* ratios between 0.05/0.95 and 0.35/0.65, the cost of the single-server (=blue) and multi-server configuration (=red) rise fast. An item has to be read before it can be changed. Therefore, *write* cost follow *read* and are upper bounded by the *read* cost in the single-server case. That is not the case in the multi-server solution. If an item is already available at the service instance a client is connected to, the item is retrieved from the local instance. An item is not available at a local service instance if no client

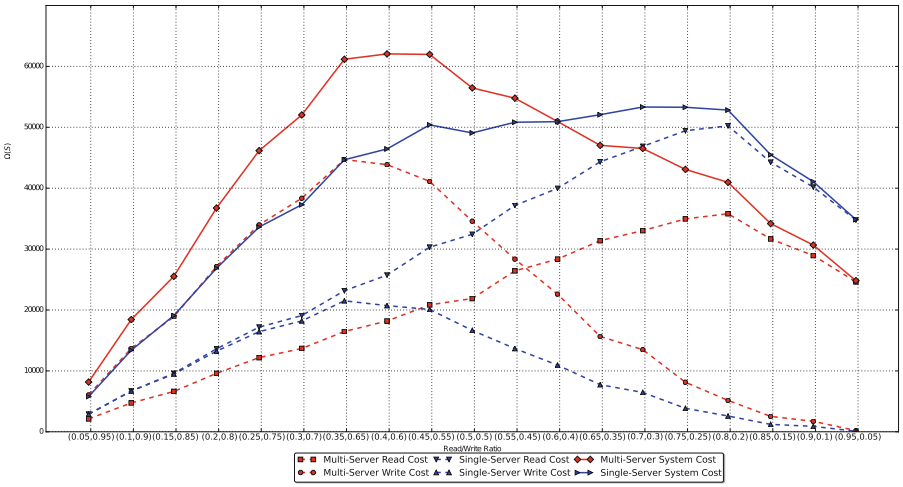

Fig. 3. Cost for different read/write profiles (Color figure online)

previously read it *or* no other client in the same context inside the *entire* network changed it. If another client would have changed it, the update was propagated. In our simulation, the multi-server cost are clearly above the single-server approach in the range of *read*/*write* ratios between 0.35/0.65 to 0.5/0.5. From 0.5/0.5, the multi-server system cost decrease. At the ratio 0.75/0.25, our self-organized system has its optimal working point. Prior to the working point, line 10 in Algorithm 1 assures cost slightly above the single-server approach. Until ratio 0.6/0.4, Algorithm 1 enforces the single-server solution. Winners of the distributed auction to serve the context can not afford the cost shares. Therefore, clients request and write data directly to the single server. After that break-even point, service instances start from being able to pay their cost shares to receive updates and our mechanism causes the convergence to the multi-server configuration. At the working point, the service can be delivered at the same cost a single-server approach would have. However, the users have a lower latency. After the break-even point, multi-server configuration outruns the single-server approach.

Though, at the ratio 0.6/0.4, our algorithm violates the cost-recovery property for time τ. That is a disadvantage for service instances with a low number of clients. Those easily run in the situation where they can not cover their cost shares anymore. The mechanism in Algorithm 1, line 10 shuts down those service instances. Their revenue function stabilized at *read*/*write* ratios of 0.75/0.25. Therefore, our self-organized algorithm fits well to services, that regularly have 75% *consumers* and 25% *producers*.

6 Conclusion

This paper presents a cross-monotonic cost sharing scheme that solves the Service Placement Problem with *information producers* and *information consumers*. It is applicable for wireless as well as for IoT infrastructures. Our approach transforms a global cost function (Eq. 1) into a local optimization problem (Eq. 2) that is optimized by a self-organized algorithm. Thus, our algorithm runs and decides in each node solely based on local knowledge. Our approach shows a favorable working point at a *read*/*write* ratio of 0.75/0.25. The service can be delivered at the same cost a single-server approach achieves while the transfer of the up-to-date data is achieved at a considerably lower delay.

Self-organized systems benefit from cross-monotonic cost-sharing schemes. Cross-monotonicity assures that newly entering agents can not increase the cost shares of other nodes. That makes it ideal for service providers because a joining agent can not increase the cost shares of other agents. Our mechanism automatically shuts down entities that suffer from a lack of *consumers* and therefore provides an efficient mechanism in dealing with an unbalance between *information consumers* and *information providers*. It is a derivate of the cross-monotonic Multicast Game. However, an agents revenue changes over time and if the revenue becomes negative, our mechanism shuts down the agent. The mechanism allows agents to violate the cost-recovery property for a time τ.

In our future work, we will give proof on the competitiveness and α-cost-recovery in dependence of τ and real-world client-profiles. Furthermore, we will compare our scheme to connected facility location algorithms that perform an offline optimization. Additionally, the bidding mechanism for the approximation of the K-Median problem has to be compared to Pál et al. [8] Primal/Dual mechanism.

References

1. Bauer, C., Dey, A.K.: Considering context in the design of intelligent systems: current practices and suggestions for improvement. J. Syst. Softw. **112**, 26–47 (2016)
2. Jaffal, A., Kirsch-Pinheiro, M., Le Grand, B.: Unified and conceptual context analysis in ubiquitous environments. In: International Academy, Research, and Industry Association (IARIA), vol. 8, pp. 48–55 (2014)
3. Jaffal, A., Le Grand, B., Kirsch-Pinheiro, M.: Refinement strategies for correlating context and user behavior in pervasive information systems. Procedia Comput. Sci. **52**, 1040–1046 (2015)
4. Krause, A.: Submodular function optimization (2008). https://las.inf.ethz.ch/sfo/
5. Najar, S., Pinheiro, M.K., Souveyet, C.: Service discovery and prediction on pervasive information system. J. Ambient Intell. Hum. Comput. **6**(4), 407–423 (2015)
6. Naqvi, S.N.Z., Ramakrishnan, A., Preuveneers, D., Berbers, Y.: Walking in the clouds: deployment and performance trade-offs of smart mobile applications for intelligent environments. In: International Conference on Intelligent Environments (IE13), 16–19 July 2013, Athens, Greece (2013)

7. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, New York (2007)
8. Pál, M., Tardos, É.: Group strategy proof mechanisms via primal-dual algorithms. In: 44th Symposium on Foundations of Computer Science (FOCS 2003), Proceedings, 11–14 October 2003, Cambridge, MA, USA, pp. 584–593 (2003)
9. Ramakrishnan, A., Preuveneers, D., Berbers, Y.: Enabling self-learning in dynamic and open IOT environments. Procedia Comput. Sci. **32**, 207–214 (2014)
10. Tazari, S.: Cross-monotonic cost-sharing schemes for combinatorial optimization games: a survey: course Project, CPSC 532A Multiagent Systems. University of British Columbia, Vancouver, Canada (2005)
11. Wu, C., Xu, Y., Chen, Y., Lu, C.: Submodular game for distributed application allocation in shared sensor networks. In: Proceedings of the IEEE INFOCOM 2012, 25–30 March 2012, Orlando, FL, USA, pp. 127–135 (2012)