A Space-Based Generic Pattern for Self-Initiative Load Clustering Agents

Eva Kühn, Alexander Marek, Thomas Scheller, Vesna Sesum-Cavic, Michael Vögler, and Stefan Craß

Vienna University of Technology, Institute of Computer Languages
Argentinierstr. 8, Vienna, Austria
{eva,amarek,ts,vesna,mvoegler,sc}@complang.tuwien.ac.at

Abstract. Load clustering is an important problem in distributed systems, which proper solution can lead to a significant performance improvement. It differs from load balancing as it considers a collection of loads, instead of normal data items, where a single load can be described as a task. Current approaches that treat load clustering mainly lack of provisioning a general framework and autonomy. They are neither agent-based nor configurable for many topologies. In this paper we propose a generic framework for self-initiative load clustering agents (SILCA) that is based on autonomous agents and decentralized control. SILCA is a generic architectural pattern for load clustering. The SILCA framework is the corresponding implementation and thus supports exchangeable policies and allows for the plugging of different algorithms for load clustering. It is problem independent, so the best algorithm or combination of algorithms can be found for each specific problem. The pattern has been implemented on two levels: In its basic version different algorithms can be plugged, and in the extended version different algorithms can be combined. The flexibility is proven by means of nine algorithms. Further contributions are the benchmarking of the algorithms, and the working out of their best combinations for different topologies.

Keywords: Agents, Load Clustering, Load Balancing, Coordination, Tuple Space.

1 Introduction

Clustering or *cluster analysis* is a method of unsupervised learning and a technique for the analysis of statistical data. It is used in many fields, including data mining, machine learning and information retrieval. Clustering deals with the problem of grouping a collection of observations into smaller subsets, so called clusters. A cluster therefore consists of elements which are similar in some way and dissimilar to elements that belong to other clusters. The greater the similarity within a cluster and the greater the difference between the clusters, the better or more distinct is the clustering.

Load clustering, as the name already states, deals with the clustering of work loads in a computer system. It is strongly related to *load balancing*, which is a

M. Sirjani (Ed.): COORDINATION 2012, LNCS 7274, pp. 230–244, 2012.

methodology to distribute load among multiple computers to achieve an optimal utilization of resources. The difference between the two is that load clustering tries to make further optimizations of the load distribution based on the content of the load items: A single load item can be described as a task that consists of several attributes (e.g. a certain priority), has a payload, a dynamic life cycle and is handled by a computer or processor. The goal of load clustering is to cluster loads not only on the basis of simple attributes but also take into consideration the payload, as well as the dynamic and therefore changing status of the system load. Load clustering is a derived form of simple data clustering. Its main goal is to increase performance, by allowing a worker in a computer system to process not only a single load at once but a cluster of loads which are similar and therefore easier and faster to process.

Since load clustering systems are complex and need to react to various factors, it is important that they are self-organizing and adaptive, so that they can flexibly adapt to dynamically changing loads and resources. Different algorithms and configurations are needed to satisfy different kinds of load clustering scenarios, so a framework is needed that allows the comparison of algorithms and fine-tuning of their behavior to achieve optimal performance results. There are currently no frameworks with the needed degree of flexibility to satisfy these requirements. Existing frameworks are specialized for data clustering, not load clustering. Moreover they follow no agent based approach, hence need a central coordinator. This makes the system more prone to errors since that coordinator is a single point of failure.

In this paper we present a load clustering framework, that provides the possibility for plugging and benchmarking different clustering algorithms. It is based on autonomous agents with decentralized control and a blackboard based communication mechanism.

According to this specification the framework is called *Self-Initiative Load Clustering Agents* (in short SILCA). The design of SILCA is based on [18], which is a generic architectural pattern for load balancing that consists of several sub-patterns that can be composed to solve different problem scenarios.

In section 2 we review existing work about clustering in general and load clustering in particular. In section 3 we present our load clustering approach and explain the different patterns that are part of it. To show the validity of our approach we evaluate it with several different load clustering algorithms which are presented in section 4. The results of this evaluation are shown in section 5, where we present benchmarks for each load clustering algorithm.

2 Related Work

Until today, a lot of research has been done on the subject of clustering and a broad range of solutions around that problem has evolved spanning different problem domains.

An interesting problem domain is search clustering. Carrot2 [30] is an open source search clustering engine allowing for automatically clustering collections of search results or document abstracts into thematic categories. Hence it

supports data clustering, not load clustering. Furthermore, Carrot2 does not allow for plugging in different clustering algorithms.

Other popular domains are data mining and analysis. There exists a broad range of open source and proprietary solutions: KNIME (Konstanz Information Miner) [3] is an open source data analysis, reporting and integration platform mainly used in pharmaceutical research. Proprietary products are STATISTICA[1] by Statsoft, SPSS Modeler[2] by SPSS Inc. and SAS[3] by SAS Institute Inc.

Another domain is machine learning. WEKA [14] (Waikato Environment for Knowledge Analysis) for example is an open source software for machine learning and supports several standard data-mining tasks like data preprocessing, clustering and classification. RapidMiner, formerly YALE (Yet Another Learning Environment) [21], is another open source machine learning environment with data-mining and clustering capabilities. Shogun [27] is an open source software toolbox focusing on kernel machines such as support vector machines for regression and classification problems. It was designed for bioinformatics applications and is therefore capable of processing datasets with up to 10 million samples. Orange [8] is another machine learning software suite designed for bioinformatics, coming with a visual front-end allowing for performing data analysis, -mining and visualization.

However, none of the mentioned software solutions supports all features we require: SPSS Modeler and SAS do not allow for the plugging of other algorithms, and the others are specialized on data clustering whereas SILCA aims to provide a framework for load clustering. Additionally, while most of these tools support extensibility through scripts, none of them follows a framework approach.

Frameworks in the area of data mining and analysis are jHepWork [6] and ELKI (Environment for DeveLoping KDD-Applications Supported by Index-Structures) [1]. jHepWork is a free data-analysis framework designed for scientists, engineers and students aiming to create a data-analysis environment based on open-source packages to create a tool that is competitive to commercial programs. However, it does not allow for benchmarking algorithms. ELKI is written in Java and allows for combining arbitrary algorithms, distance functions and indexes in order to evaluate and benchmark these combinations.

ELKI offers framework abstraction, the pluggability of algorithms and ability to benchmark those, but it does not follow a decentralized, agent-based approach, which is one of the main aims of SILCA. Moreover, to obtain the best performance results, it must be possible to fine-tune algorithms by changing related parameters and swapping similarity functions, which is not supported by ELKI.

SILCA follows a similar approach as introduced in [18] where a generic pattern for a load balancing framework is proposed, allowing for plugging and benchmarking different load balancing algorithms in different configurable settings and therefore easing the selection of the best algorithm for a specific problem scenario.

[1] http://www.statsoft.com/
[2] http://www-01.ibm.com/software/analytics/spss/
[3] http://www.sas.com/

3 Load Clustering Pattern

The objective of SILCA is the design of general patterns that abstract the problem of load clustering. Patterns are re-usable building blocks that can be composed towards solutions for certain problems like in our case the load clustering scenario. The SILCA *framework* is built according to these patterns to provide the needed flexibility.

The main requirement on the SILCA design is support of decentralized control so that the system can flexibly react on dynamically changing loads and resources, which is a basic condition for a self-organized, adaptive system. Moreover, a peer-to-peer system is less vulnerable. This means the avoidance of a central coordinator and leads to a software architecture design based on autonomous agents. Such an agent is an autonomic software component [10] that is self-responsible to be up and running, implements a reactive and continuous behavior, and can dynamically join and leave.

The blackboard based architectural style supports very well communication and synchronization between many independent, distributed software components, especially if they carry out computations where a complex task must be divided into smaller ones [2]. Therefore the SILCA design is based on a secure tuple space based middleware [7] that has proven useful in agent based use cases [17]. We assume the possibility to add reactive behavior [9] which especially enables notifications in near time as well as dynamic policies that are triggered by the arising of events, and the support of transactions with specifiable timeouts.

A second requirement on SILCA is the flexible exchange of different algorithms (see section 4) simply through "plugging" in order to gain a testbed for the comparison and evaluation of best solutions for certain problem scenarios. This is achieved by means of a component based design of the agents.

In all patterns, shared spaces hold the information produced by agents, as well as all events on which agents react. Pattern composition is carried out in that several agents access the same space and agree about its entries' structures, semantics and coordination principles.

The sub-patterns of SILCA and their implementation and composition towards a load clustering framework that can be configured for many different network topologies are explained in the following.

3.1 Local Node Pattern

The local node pattern has many similarities with local load balancing [25] as on one single computer site there is no load to be distributed or clustered, yet. Its main purpose is to model the autonomous agents as independent workers that actively compete for work. Worker agents register themselves at a load space. Clients write work into the load space which triggers the workers by means of aspects to take the load in chunks, process it and write the result into a result space (Figure 1) where the clients will pick them up eventually. If a worker fails,

another one takes over its work. This is achieved by using the same transaction to take the load and write the result back. This transaction possesses a timeout and if it expires, the entire action of the worker is rolled back, the locks on the taken entries are released and another worker can proceed.

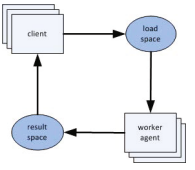

Fig. 1. Local node pattern

3.2 Arbiter Pattern

The arbiter pattern implements the clustering activation policy which determines whether load shall be shifted from the local node to a cluster. This can e.g. be configured by a parameter that specifies a certain threshold for loads: If this amount is exceeded and if not enough workers are available, load is taken from the load space and written to a clustering space (Figure 2). Comparable to worker agents, the arbiter agents are also implemented as space aspects that are activated every time when new load arrives, and in addition also when load is removed, or a new worker is de/registered. The arbiter also reacts on clustered loads that it receives via the clustering space and moves it from there to the local load space.

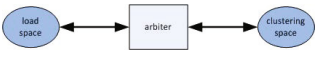

Fig. 2. Arbiter pattern

3.3 Clustering Pattern

The clustering pattern consists of clustering agents that execute a clustering strategy to distribute the load in the network. They access clustering spaces and clustering agent spaces. Figure 3 shows this pattern with one clustering space and one clustering agent space. The latter space holds information like neighbor nodes, pheromones etc. that the clustering agents need to collaborate with each other by executing a particular algorithm (see section 4).

3.4 Pattern Composition

The three described patterns can now be used to build up arbitrary load clustering patterns. The composition of the local node, arbiter and clustering pattern

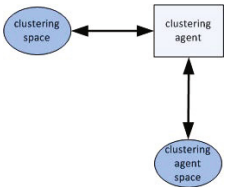

Fig. 3. Clustering pattern

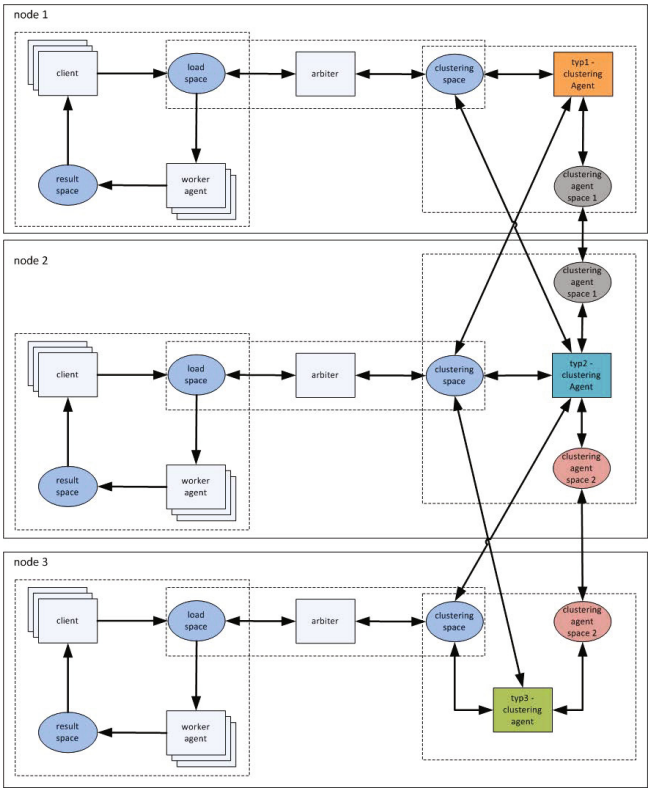

Fig. 4. Pattern composition example

forms the Basic SILCA pattern. The Extended SILCA pattern is the composition of several Basic SILCA patterns. A load clustering agent may interact with many arbiters and other load clustering agents in that it accesses multiple clustering and clustering agent spaces. Depending on the compositions of the arbiter and load clustering patterns via their shared spaces, different logical network overlay topologies arise. Figure 4 shows e.g. three single nodes that form a chain topology. Moreover, the load clustering agents might execute different algorithms within the same framework setting.

The space-based pattern approach leads to highly flexible agent coordination and an agile software architecture that is resistant against changing requirements concerning new policies and algorithms. Its advantages are a loose coupling of the collaborating agents through asynchronous communication, general abstraction of the load clustering problem, and modularization into re-usable sub-patterns.

4 Algorithms

Since many different algorithms cope with the clustering problem, we chose some of the well known and widely used clustering algorithms: Hierarchical, K-Means, Fuzzy C-Means, Genetic K-Means and Ant K-means.

As classification or statistical classification can also be seen as a supervised form of clustering, where observations get assigned into classes according to a given training set, we will also use some algorithms which cope with the classification problem, to demonstrate the agility of SILCA: K-Nearest Neighbor, Decision Tree, Ant-Miner, cAnt-Miner.

These 9 algorithms are implemented and benchmarked with SILCA, since they are well documented and implementations of them are already tested and available. The focus here is not the choice of the algorithms, but to prove the concept of SILCA with the help of them.

Hierarchical clustering is used to build a hierarchy of clusters. There are two types to build a hierarchy. The first one is the **Agglomerative** [12] approach (bottom up) where each data point is assigned to one cluster and pairs of clusters get merged to build the hierarchy. The second one is the **Divisive** [13] approach (top down), where all data points are captured in one cluster and this cluster is recursively split to build the hierarchy.

K-Means [15,28] is one of the simplest clustering algorithms. K-Means uses k centriods (one for each cluster), which get placed far away from each other. According to the location of the centriods each data point from the given data set is associated to the cluster which has the nearest centriod. In the next step k new centriods get calculated in the barycenters of the previously generated groupage. Now the data points get reassigned according to new centroids. The step of the recalculation of centriods and the reassignment of data points is performed in a loop. As a result of this loop the centriods change their location as long the centriods don't move anymore.

Fuzzy C-Means [11,4] is an adapted version of the K-Means algorithm, that allows data items to belong to more than one cluster. The Fuzzy C-Means algorithm almost works the same way as the K-Means algorithm, but with the small difference that each data item has some kind of parameter that indicates the degree of membership to a certain cluster. For each recalculation of the k centriods, the degree of membership of each data item is updated. At the end of the algorithm the degree of membership parameter can be used to place the data item in the best cluster.

The **Genetic K-Means** [16] algorithm is a mixture of the classical K-Means algorithm and an algorithm that follows Darwin's theory of evolution. The

algorithm uses an initial population of clusters, where data items get placed randomly, and the clustering gets evolved over several generations. At each generation phase, every cluster gets evaluated and fresh clusters get generated with two genetic operations: crossover and mutation. The crossover operation randomly selects a location at a cluster and concatenates two clusters at this crossover point with each other to generate new clusters. The mutation operation brings disturbance in the crossover operation by inverting some elements during the regeneration process. This operation provides diversity and prevents stagnation.

Ant K-Means [26,19,29] combines the classical K-Means algorithm with an ant colony optimization. The principle of ant colony optimization is a pheromone trail which is used by real ants to communicate with each other. When an ant follows a certain trail it leaves a specific amount of pheromones. The more ants follow this trail, the more pheromones are placed and therefore this trail becomes more attractive for other ants, which also obtains the shortest route. Now this behaviour is used in the clustering domain to produce an optimal assignment of a set of observations to several clusters. The algorithm uses R agents (ants) to build the solution. To represent the pheromone trail a so called pheromone matrix τ is used, where a pheromone value τ_{ij} stands for the pheromone concentration of observation i associated to cluster j. At any iteration of the algorithm, each agent develops a trail solution by using the pheromone matrix to produce an optimal clustering. After this step R trail solutions are produced, a local search is performed to improve the solutions and the pheromone matrix gets updated. According to the updated pheromone matrix the previous steps are repeated for a specified number of iterations to improve the solution.

The **K-Nearest Neighbor** [20,24] algorithm is one of the simplest classification algorithms. K-Nearest Neighbor classifies an object according to its k nearest neighbors, where k is a positive (small) number. These neighbors are taken from the given training set where the correct classification of the objects is already known.

Decision Tree [5] learning is a commonly used method in data mining. This algorithm uses a tree structure that consists of leaves which represent the classifications and branches which describe the conjunctions of the observations' attributes that lead to the classification. To construct the tree the algorithm chooses an attribute of the data set that efficiently splits the set into two subsets which are classified by one class or the other. This step gets recursively repeated for the sub-sets as long as there is a split of the sub-set possible.

Ant-Miner [23] and **cAnt-Miner** [22] are classification methods that use an ant colony optimization approach. These algorithms follow a sequential approach to construct a list of rules (so called classification-rules) to classify objects. The algorithms are executed several times in a loop, each time against a reduced test data set. During one cycle of the loop the ants sequentially start their rule generation phase with an empty rule set and add one term, which represents the attribute of an object, at time. The choice of adding a term depends on both a heuristic value (based on the entropy of the term) and the current pheromone level of each term. When an ant produced a rule the pheromone levels get

updated and another ant starts its run. If all possible cases of the training set are covered the loop stops and each data item of the data set is classified according to the previously retrieved classification rules.

5 Benchmarks and Evaluation

We performed benchmarks in several settings in order to demonstrate the agility of the SILCA pattern and prove that nature/swarm based enhanced algorithms can outperform several well-known algorithms. For this purpose we implemented the algorithms mentioned in Section 4.

Each test-run for one of the nine algorithms consisted of five cycles and the average was taken as result which guarantees their validity. The load tuples had the form "[taskID:12345, clientID:client1, priority:high, param:Prog1, description:'compile prog1', answerURL:url, workerType:compiler, timeout:200]" where "param" refers to a real and compilable Java class file. Each load tuple had a size of 5 to 10 kB. The comparison was done based on the given attributes and the similarities among the source code of the Java class file. All tests were executed on a cluster of four machines with 2*Quad AMD 2.0 GHz CPUs and three GB of RAM. Additionally we used three test settings for Basic SILCA and one for Extended SILCA.

5.1 Basic SILCA Benchmarks

The **first basic test setting** (Figure 5) uses only one worker to investigate how good a particular clustering-algorithm can find similarities among the load and how fast one worker can process that load.

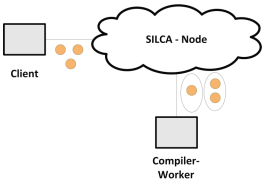

Fig. 5. Setting with 1 worker

In the **second and third basic test setting** (Figures 6 and 7), we added additional workers to investigate how the clustering results change if the worker-type is also taken into consideration and how the number of workers affect the clustering performance.

For each of the three basic test settings, each algorithm was benchmarked with 10, 20 and 50 loads. The results are shown in the figures 8, 9 and 10. Note that increasing the number of workers does not decrease execution time, because

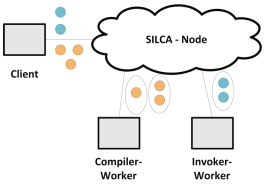

Fig. 6. Setting with 2 workers

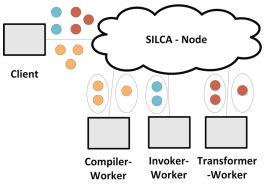

Fig. 7. Setting with 3 workers

each worker is responsible for a separate load type. In essence, increasing the number of workers also increases clustering complexity and thus execution time.

Using the absolute execution time as metric for the benchmarks, the Hierarchical algorithm shows the best results for all test-settings. Not only is it fast in constructing clusters, but also performs distinct and therefore good clustering. Additionally, its execution time is nearly constant in all three test-settings. More surprisingly is the fact that the biological enhanced Ant K-Means algorithm performs better than K-Means and quite equal to Fuzzy C-Means. The worst performing algorithm is Genetic K-Means, because of the algorithmic complexity of the genetic approach. According to the results, the classification algorithms also perform well on the given test settings. All of them deliver a good grouping of the loads and are able to keep up with the clustering algorithms. The best of them is the Ant-Miner algorithm due to the result of the fast rule generation phase and a good and distinct classification. Ant-Miner performs 2% faster than Decision Tree, 5% faster than cAnt-Miner and 14% faster than K-Nearest Neighbour. Decision Tree is not as good as Ant-Miner, since the generation of the tree is not as fast as the rule generation phase of Ant-Miner. The biologically enhanced cAnt-Miner does not outperform the well-known Decision Tree algorithm due to the fact that it has a pretty slow rule generation phase. The K-Nearest Neighbour algorithm performed worst during the benchmarks, since this algorithm strongly depends on the amount of neighbours to choose the correct class and therefore produced a worse classification for some test-settings. Nevertheless we have to mention that the training step of the classification algorithms was not included into the absolute execution time.

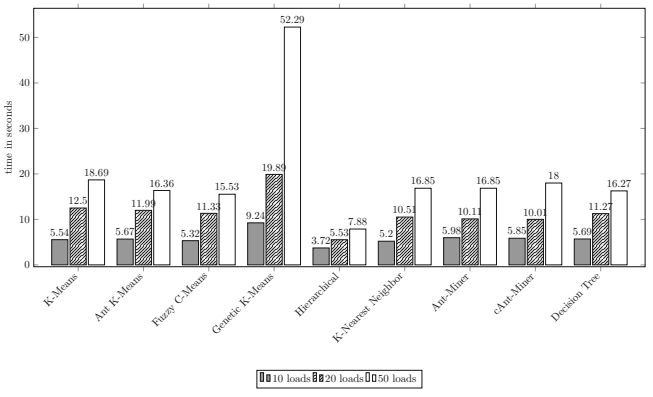

Fig. 8. Comparison of algorithm results for one worker

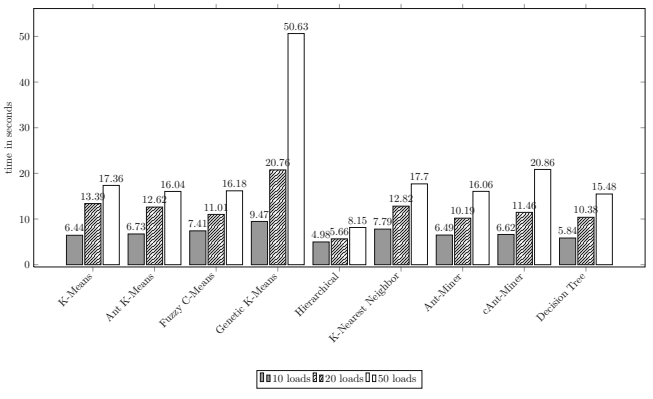

Fig. 9. Comparison of algorithm results for two workers

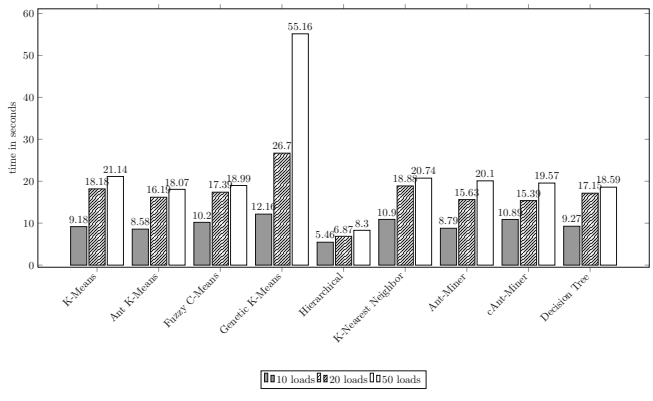

Fig. 10. Comparison of algorithm results for three workers

5.2 Extended SILCA Benchmarks

The extended SILCA pattern makes use of SILCAs composability and hence allows for combining different algorithms within one collaborative clustering approach. To prove this we created a test setting that allows for combining two algorithms at a time (Figure 11).

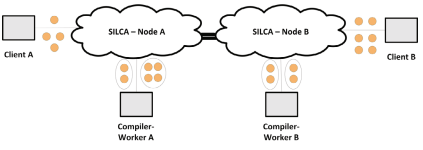

Fig. 11. Test Setting for extended SILCA benchmarks

In each benchmark, two clients assign 20 loads each, so in total 40. The metric used in these benchmarks is the absolute execution time. According to the obtained results (Figure 12), it can be seen that the combination of the Hierarchical algorithm with any other, except the Genetic K-Means algorithm, leads to a good execution time. The best result is delivered by the combination of the Hierarchical and Fuzzy C-Means algorithm, which is obvious since both perform well in the basic benchmarks. Also the combination of Ant K-Means with Hierarchical and Fuzzy C-Means with Ant K-Means produces pretty good results. Any combination with Genetic K-Means, compared to the other combinations, is extremely slow (taking up to 295% more time than the fastest combination), which is also foreseeable since the genetic aspect leads to a big performance lack in these test settings.

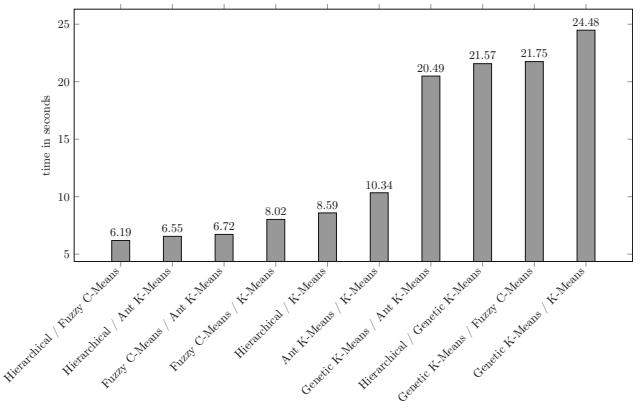

Fig. 12. Comparison of algorithm combinations in extended SILCA

The aforementioned benchmark settings are clearly kept simple and are not meant to be representative for real-world scenarios. Yet we claim that they are sufficient as a proof of concept and to demonstrate that increasing workers and worker types can increase execution time, and mixing different clustering algorithms can lead to better performance

6 Conclusions

In this paper, we presented a generic framework for self-initiative load clustering agents (SILCA), which is based on autonomous agents that communicate and operate in a peer-to-peer manner, and decentralized control. SILCA is a composable and agile software architecture pattern for load clustering that has been implemented on two levels: basic and extended. SILCA is problem independent and allows for plugging different clustering and classification algorithms (both intelligent and unintelligent). Basic SILCA consists of several sub-patterns, implemented in a space-based architectural style, which allows decoupling of the agents and guarantees their autonomic behavior. This allows finding the best algorithm for each specific problem. In extended SILCA several Basic SILCA nodes are connected via shared spaces towards arbitrary network topologies, which supports the plugging of combinations of different algorithms. Further contributions include benchmarking of the algorithms, and finding their best combinations for different topologies. The following clustering and classifying algorithms have been implemented and benchmarked on a cluster of 4 machines through 4 different test settings: K-Means, Ant K-Means, Fuzzy C-Means, Genetic K-Means, Hierarchical Clustering, K-Nearest Neighbor, Ant-Miner, cAnt-Miner, and Decision Tree. The absolute execution time was used as metric for the benchmarks. The preliminary results show the following: From the group of clustering algorithms, Hierarchical Clustering obtained the best results, whereas from the group of classification algorithms the Ant-Miner algorithm was the best which is the result of the fast rule generation phase and a good and distinct classification. In the extended SILCA, several combinations were benchmarked. The combination of the Hierarchical algorithm with any other, except the Genetic K-Means algorithm, leads to a good execution time. The best result was delivered by the combination of the Hierarchical and Fuzzy C-Means algorithm (both performed well in the Basic SILCA benchmarks, too). The unintelligent Hierarchical Clustering showed the best results in a small network with only one client that supplies load. For large and more complex networks, an intelligent approach with an appropriate similarity function will help. The similarity function is a crucial issue on which the quality of obtained clusters depends, so an intelligent approach should provide an improvement in results: both quantitatively - the absolute execution time, and qualitatively – the quality of obtained clusters. Future work will include the following issues: Plugging of new intelligent clustering algorithms based on bee intelligence and slime mold behavior, composition of load clustering and load balancing, and benchmarking on networks with other topologies and of larger dimensions.

Acknowledgments. The work is partly funded by the Austrian Government under the program BRIDGE (Brückenschlagprogramm der FFG), project 827571 AgiLog – Komplexitätsreduzierende Middleware-Technologien für Agile Logistik and under the program FFG FIT-IT (Forschung, Innovation und Technologie für Informationstechnologien), project 825750 Secure Space – A Secure Space for Collaborative Security Services.

References

1. Achtert, E., Kriegel, H.-P., Zimek, A.: ELKI: A Software System for Evaluation of Subspace Clustering Algorithms. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 580–585. Springer, Heidelberg (2008)
2. Avgeriou, P., Zdun, U.: Architectural patterns in practice. In: Longshaw, A., Zdun, U. (eds.) EuroPLoP, pp. 731–734. UVK - Universitaetsverlag Konstanz (2005)
3. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., Wiswedel, B.: KNIME: The konstanz information miner. In: Preisach, C., Burkhardt, H., Schmidt-Thieme, L., Decker, R. (eds.) Data Analysis, Machine Learning and Applications. Studies in Classification, Data Analysis, and Knowledge Organization, pp. 319–326. Springer, Heidelberg (2008)
4. Bezdek, J.C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Kluwer Academic Publishers, Norwell (1981)
5. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees, 1st edn. Chapman and Hall/CRC (January 1984)
6. Chekanov, S.: Hep data analysis using jhepwork and java. In: Proceedings of the Workshop HERA and the LHC, 2nd Workshop on the Implications of HERA for LHC Physics (2008)
7. Craß, S., Kühn, E.: Coordination-based access control model for space-based computing. In: 27th Annual ACM Symposium on Applied Computing (2012)
8. Demšar, J., Zupan, B., Leban, G., Curk, T.: Orange: From Experimental Machine Learning to Interactive Data Mining. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS (LNAI), vol. 3202, pp. 537–539. Springer, Heidelberg (2004)
9. Denti, E., Omicini, A.: An architecture for tuple-based coordination of multi-agent systems. Softw. Pract. Exper. 29, 1103–1121 (1999)
10. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Trans. Auton. Adapt. Syst. 1, 223–259 (2006)
11. Dunn, J.C.: A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. Journal of Cybernetics 3(3), 32–57 (1973)
12. Gowda, K.C., Krishna, G.: Agglomerative clustering using the concept of mutual nearest neighbourhood. Pattern Recognition 10(2), 105–112 (1978)
13. Gowda, K.C., Ravi, T.V.: Divisive clustering of symbolic objects using the concepts of both similarity and dissimilarity. Pattern Recognition 28(8), 1277–1282 (1995)
14. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. 11, 10–18 (2009)
15. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. 31, 264–323 (1999)
16. Krishna, K., Narasimha-Murty, M.: Genetic K-means algorithm. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) 29(3), 433–439 (1999)

17. Kühn, E., Mordinyi, R., Keszthelyi, L., Schreiber, C.: Introducing the concept of customizable structured spaces for agent coordination in the production automation domain. In: The Eighth International Conference on Autonomous Agents and Multiagent Systems, AAMAS, May 10-15, pp. 625–632 (2009)
18. Kühn, E., Sesum-Cavic, V.: A Space-Based Generic Pattern for Self-Initiative Load Balancing Agents. In: Aldewereld, H., Dignum, V., Picard, G. (eds.) ESAW 2009. LNCS, vol. 5881, pp. 17–32. Springer, Heidelberg (2009)
19. Kuo, R.J., Wang, H.S., Hu, T.L., Chou, S.H.: Application of ant k-means on clustering analysis. Comput. Math. Appl. 50, 1709–1724 (2005)
20. Mhamdi, F., Elloumi, M.: A new survey on knowledge discovery and data mining. In: RCIS, pp. 427–432 (2008)
21. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: YALE: rapid prototyping for complex data mining tasks. In: KDD 2006: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 935–940. ACM, New York (2006)
22. Otero, F.E.B., Freitas, A.A., Johnson, C.G.: Handling continuous attributes in ant colony classification algorithms. In: CIDM, pp. 225–231. IEEE (2009)
23. Parpinelli, R., Lopes, H., Freitas, A.: Data Mining with an Ant Colony Optimization Algorithm. IEEE Trans. on Evolutionary Computation, Special Issue on Ant Colony Algorithms 6(4), 321–332 (2002)
24. Phyu, T.N.: Survey of classification techniques in data mining. In: Proceedings of the International Multi Conference of Engineers and Computer Scientists 2009, IMECS 2009, Hong Kong, March 18-20. Lecture Notes in Engineering and Computer Science, vol. I, pp. 727–731. International Association of Engineers, Newswood Limited (2009)
25. Šešum-Čavić, V., Kühn, E.: Chapter 8 Self-Organized Load Balancing through Swarm Intelligence. In: Bessis, N., Xhafa, F. (eds.) Next Generation Data Technologies for Collective Computational Intelligence. SCI, vol. 352, pp. 195–224. Springer, Heidelberg (2011)
26. Shelokar, P.S., Jayaraman, V.K., Kulkarni, B.D.: An ant colony approach for clustering. Analytica Chimica Acta 509(1) (2004)
27. Sonnenburg, S., Rätsch, G., Henschel, S., Widmer, C., Behr, J., Zien, A., de Bona, F., Binder, A., Gehl, C., Franc, V.: The SHOGUN Machine Learning Toolbox. Journal of Machine Learning Research (2010)
28. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining, 1st edn., ch. 8. Addison-Wesley Longman Publishing Co., Inc., Boston (2005)
29. Tiwari, R., Husain, M., Gupta, S., Srivastava, A.: Improving ant colony optimization algorithm for data clustering. In: Proceedings of the International Conference and Workshop on Emerging Trends in Technology, ICWET 2010, pp. 529–534. ACM, New York (2010)
30. Weiss, D.: A Clustering Interface For Web Search Results In Polish And English. Master's thesis, Poznan University of Technology, Poland (2001)