Proactive Container Auto-scaling for Cloud Native Machine Learning Services

David Buchaca
Barcelona Supercomputing Center
Universitat Politecnica de Catalunya
david.buchaca@bsc.es

Josep LLuis Berral
Barcelona Supercomputing Center
Universitat Politecnica de Catalunya
josep.berral@bsc.es

Chen Wang
IBM Research
Yorktown Heights, NY
Chen.Wang1@ibm.com

Alaa Youssef
IBM Research
Yorktown Heights, NY
asyousse@us.ibm.com

***Abstract*—Understanding the resource usage behaviors of the ever-increasing machine learning workloads are critical to cloud providers offering Machine Learning (ML) services. Capable of auto-scaling resources for customer workloads can significantly improve resource utilization, thus greatly reducing the cost. Here we leverage the AI4DL framework [1] to characterize workload and discover resource consumption phases. We advance the existing technology to an incremental phase discovery method that applies to more general types of ML workload for both training and inference. We use a time-window MultiLayer Perceptron (MLP) to predict phases in containers with different types of workload. Then, we propose a predictive vertical auto-scaling policy to resize the container dynamically according to phase predictions. We evaluate our predictive auto-scaling policies on 561 long-running containers with multiple types of ML workloads. The predictive policy can reduce up to 38% of the allocated CPU compared to the default resource provisioning policies by developers. By comparing our predictive policies with commonly used reactive auto-scaling policies, we find that they can accurately predict sudden phase transitions (with an F1-score of 0.92) and significantly reduce the number of out-of-memory errors (350 vs. 20). Besides, we show that the predictive auto-scaling policy maintains the number of resizing operations close to the best reactive policies.**

***Keywords*-Cloud Native; Machine Learning Service; Container; Auto-scaling;**

I. Introduction

Containers provide a lightweight, fast and isolated infrastructure to run applications. The container-based environment, namely cloud-native, is becoming the de facto standard for deploying services in the cloud, especially for Cloud Machine Learning Services [2], [3], [4]. Machine learning services deserve special attention when managing resources as their resource usage behaviors differ significantly from those traditional long-running services, such as web services or databases. When machine learning workloads are containerized, multiple containers running different tasks (training or inference) can co-locate on the same machine and each alternate strides of data fetching and data processing with dynamically varying resource usage. On the other hand, container orchestration system such as Kubernetes requires developers to declare a fixed size for containers to run these jobs. Thus, developers tend to over-provision resources for each container, and the cluster ends up with very low resource utilization. Auto-scaling containers vertically, according to the actual usage, would improve resource efficiency. However, they may cause out of memory (OOM) kill or Quality of Service (QoS) degradation under sudden usage increase.

Proper auto-scaling of containers vertically requires understanding the dynamics of resource usage. No universal heuristics can work for different types of workloads. Machine learning workload usually includes routines from data loading to computation, showing random spikes in memory and CPU usage, which cannot be captured by classical time-series algorithms. Transitions of those behaviors can be abrupt, complex, and shifting across different containers. Studying phase transitions across multiple containers running different models and tasks can help learn in general how phases in machine learning workload transit; however, traditional time series prediction methods cannot achieve this goal. Container auto-scaling itself is not an easy task. A tight auto-scaling can lead to more frequent resizing operations, while a slack auto-scaling needs fewer resizing operations but results in more resources over-provisioned. Reactive approaches[5], [6], in general, can only respond to a gradual change and usually fail to foresee sudden behavior changes.

In this paper, we employ CRBMs and clustering techniques in [1] to discover phases and extend it with an incremental mechanism to apply to more kinds of workload, e.g., voice recognition services, machine learning training, and inference services. We propose to use MLPs to learn sequences of phases discovered from multiple containers running different types of workloads to predict sudden behavior changes. Given the prediction of phases, we apply a proactive policy to auto-scale the container size according to the phase prediction. To reduce the number of unnecessary resizing actions, we only scale containers if the forecast foresees significant behavior changes.

To summarize, we find that CRBM based workload characterization is suitable for modeling phase behaviors, including sudden spikes or random burstiness. The proposed iterative approach can incrementally discover new phases for new types of workload. We also find that there exist common patterns of phase transitions. Both MLP models and LSTM models can well predict typical phase transitions. The F1-score of MLP is 0.93 for one-step prediction and is 0.79 for a four-step prediction. Compared to reactive auto-scaling policies (560 OOMs), we find that our method (20 OOMs) can prevent 96% OOM kills while reducing the over-provisioning up to 38% CPU.

II. Related Work

Various challenges in workload characterization have been addressed, including adaptive model selection[7] and adaptive time window selection[8]. [9] focuses on modeling workloads to discover patterns and classify applications, so heterogeneous workloads can be co-located to achieve higher resource utilization. [10] is the first to use Conditional Restricted Boltzmann Machines (CRBMs) and the Hidden Markov Models to discover Spark applications' profiles in an unsupervised way. [1] further applies the CRBM approach and the clustering method to identify phases in resource usage of containerized Deep Learning (DL) training workload. However, they did not fully take advantage of the predictability in dynamic resource allocation.

Workload prediction is a known challenge. [11], [12], [13]rely on such an assumption to address resource management for cloud services and applications. They mainly predict resource utilization based on the arrival of queries or clients, using predictors, including regression[12], ARMA [13], and ARIMA [11]. These models are well-known for capturing time-dependent structures, such as trends and seasonality. However, they are prone to fail when resource usage consists of phases with irregular behaviors and sudden transitions of phases. More recent work [14] uses a more advanced model, namely an ensemble of various ML models to predict future resource utilization based on job arrivals and monitored past job runs. However, these methods cannot be applied to the vertical auto-scaling of containers, as each runs a single job.

Resource auto-scaling has been an active research field for years. Many studies [15], [16], [17] focus on horizontal auto-scaling, which automatically changes the number of instances (containers or virtual machines) for a particular application when the load fluctuates. A few others focus on the vertical auto-scaling [18]. There are typically two types of approaches, reactive and proactive auto-scaling. The reactive approach usually adapts the size of the containers according to the real-time changes in resource usage, such as Kubernetes vertical pod autoscaler (VPA) [6] and [19]. Google's Autopilot system [5] adopts an advanced reinforcement learning approach to fine-tune the common heuristics to both reduce the over-provisioning slacks and the out of memory events. However, reactive strategies always have delays in the container resizing when workload behaviors change abruptly. Proactive auto-scaling [20] focuses on developing a good prediction model to resize an instance based on predictions of future resource usage. However, these prediction models can hardly capture sudden behavior changes, which commonly occur in ML workloads.

III. Methodology

The proposed methodology extends and puts into practice the profiling mechanism for behavior discovery and classification described in AI4DL [1]. AI4DL 1) monitors the principal container resources to be provisioned in production systems (i.e., CPU and Memory consumption, but also possible GPU and IO), 2) encodes these metrics to capture the dynamics over time (behavior), 3) classifies those behaviors by similarity, 4) reduces the whole execution to a sequence of "behavior phases", each with a unique resource consumption profile. In this work, we predict the future transition of phases from an ongoing execution, towards advancing that profiling information and preemptively provision containers.

A. *Overview of System Design*

Figure 1 shows our proactive container auto-scaling system. The AI4DL framework [1] collects workload traces to learn a phase discovery model, extended here with an incremental phase discovery method to accommodate more types of workload. Each container's resource consumption is modeled as a sequence of phases, each phase with its statistic profiling (e.g., maximum, average, standard deviation). A phase prediction model is trained on phase sequences to forecast phases for a running container. Once in production, new applications are passed through the phase discovery model, to obtain their current behaviors and forecast the next phases. Given the prediction for the next execution interval, proactive policies can decide how to resize the container using the predicted phase profile.

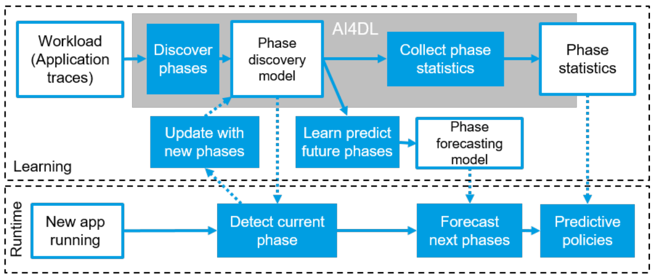


Figure 1: The proactive auto-scaling system, using AI4DL, incremental phase discovery and phase forecasting

B. *Incremental Phase Discovery*

The phase discovery process consists of the following three main elements:

Trace encoding: Telemetry from containers is encoded using a Conditional Restricted Boltzmann Machine (CRBM) [21], capable of encoding multi-dimensional time-series into a features vector, where similar inputs produce similar encoding. CRBM receives a time window of metrics, namely CPU and memory at time t, plus the history $t-1 \ldots t-d$, where d is the CRBM *delay* (time-window size as hyper-parameter). The output is a vector of size h embedding the full window trace, capturing the *dynamic* *behavior* or the resource usage for the given time window $t \ldots t-d$.

Phase identification: As CRBM-generated encoding vectors have a "similarity" property, proximity-based clustering methods like k-means are used to cluster similar behaviors, each with a similar resource usage profile. The number k can be determined during the clustering training through classical methods (e.g., observing the Square Sum Within clusters (SSW) [10]), or to be incremental.

Statistical profiling: By passing real-time container metrics through the encoding and clustering, we can identify the current phase of execution at each time step. For each phase discovered, we keep its profiling statistics to provide information to resource allocation policies.

From an initial set of workloads, we can discover their phases and phase profiles. However, when new containers from a different kind of application arrive, new behaviors are introduced into the system. When a behavior classified in a specific phase differs largely from the usual center, we should consider it as a new phase. As observed during experimentation, the CRBM does not need to be retrained when the new application shows an unidentifiable behavior, but the clustering method does. Also, we can either create a new clustering model for the application or split the divergent clusters adding the new centers to the phase identification model.

C. Phase Prediction

As previously seen in AI4DL [1], detecting the current phase of a running container can provide better estimations of resource usage than taking those from the previously observed time window. However, forecasting the next phases should be better if phase prediction is possible. We propose Multi-Layer Perceptron (MLP) with time-windows to predict the next sequence of phases given a running execution. When comparing it with other predictors, such as a Long Short-Term Memory (LSTM), we observed similar performance for our benchmarking executions. We also studied naïve forecasting rules and simpler predictors, which obtain lower accuracy with respect to MLPs and LSTMs.

Given a sequence of phases discovered from a running container in a time window $(t-d \ldots t)$, we use the MLP to produce $(t+1 \ldots t+n)$. We use the statistics for each predicted phase and decide to resize the container according to the maximum resource usage observed. The prediction model is trained on a collection of sequences of phases from completed containers. All phases use the same interval, 15 seconds, to generate a phase ID. The phase prediction task is cast as a classification problem. The classifier takes as input a time window of phase values and predicts as output the phase IDs in the next n time slots.

D. Container Auto-scaling Policies

When applying our phase-detection and phase-forecasting in container auto-scaling, we explore two different types of auto-scaling policies: reactive policies (e.g., as used in [5], [1]) and proactive policies (policies based in forecasting).

Reactive policies: such policies resize containers according to the information observed in the past. We evaluate two strategies based on time windows statistics. The first strategy makes a resizing decision according to the top 95^{th} percentile resource demands from the previous time window. The second strategy makes a resizing decision based on the 95^{th} percentile of resource demands from the phases profiles in the time window.

Proactive policies: such policies allocate resources through forecasting resource usage, i.e., predicting the next several phases and using their profiles to resize the container proactively. When a proactive policy decides how much to provision, it takes into account the "next window" of predictions. Our policy chooses the maximum of the candidate profiles for a future window. The forecasting window size becomes a hyper-parameter to study.

Finally, we need to decide how frequently to predict and resize the containers. Two strategies are available: 1) we resize the containers periodically (e.g., each N minutes), or 2) we trigger the resizing only when necessary (e.g., the need for more resources). In the IBM Cloud services, metrics can be collected every 15 seconds as a "step" where a phase can be detected (i.e., with $d = 3$, the CRBM + clustering can detect a phase with a time window of 4 steps = 1 minute). In AI4DL a periodical resizing policy is applied every 10 minutes according to their system requirement, while in AutoPilot resizing is only triggered when the difference between the resource usage and the limit exceeds a tolerance, to prevent frequent unnecessary resizing operations.

Here we apply an auto-scaling policy that 1) at each time step predicts phases in the next time window (1 minute), 2) retrieves the required CPU and Memory resources from all predicted phase profiles, and 3) determines the maximum resource demand from all predicted phase profiles (taking the percentile rule into account as mentioned above). If the predicted resource demands indicate an increase over a given tolerance (namely 10% of the current demand), the container is scaled up with the predicted resources. And when the predicted resources indicate a decrease of demand below the current limit, we scale the container down with larger tolerance to avoid some slight usage fluctuations causing frequent scaling up and down.

IV. Experiments

A. Incremental Phase Discovery

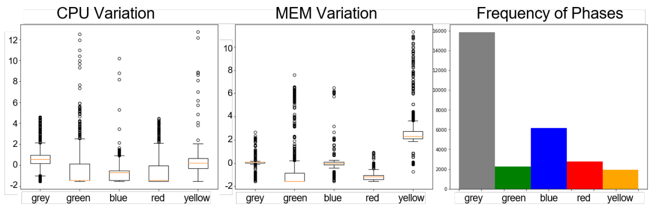
First, we evaluate the iterative approach for the incremental phase discovery by adapting phase models trained on DLaaS workloads to VR workloads, in clusters that provide the IBM Watson ML service. We use 5000 container traces from DLaaS for training and 500 for validation, and 50 traces from VR ($\times 100$ longer than DLaaS traces) for testing the model generalization and incremental updates.

A CRBM is defined by the number of inputs and the number of hidden units. The model is trained for a specified number of epochs using a gradient descent algorithm. We use the Square Sum of the Error on data reconstruction as the loss to stop learning.Let us denote by d the sliding window size, and by t the time at which the sliding window is placed $(t-d,\dots,t)$. After an exploration of the loss of CRBM, we chose the optimal delay to $d=3$.Therefore, if the sampling period in our cluster is 15 seconds, the time window used to discover phase is one minute (of 15 seconds + $3*15=45$ seconds of history). By tuning the hyper-parameters, we find the minimum loss with a learning rate $lr=10^{-3}$, 2000 epochs, and 10 hidden units. To find the different phases, we use a *k-means*, proved to work as well as other sophisticated methods. *K-means* only requires tuning the number of cluster k, selected by studying the Square Sum Within clusters (SSW) [22]. In our data, most of the SSW is explained with $k=5$.

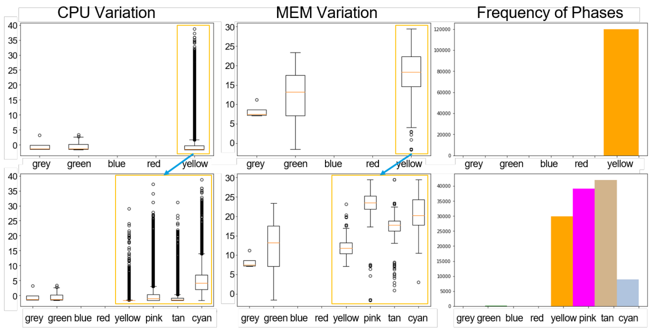
Figure 2 shows the phase discovery and detection results for both the DLaaS workload and the VR workload. We first train the model just on DLaaS container traces. Then, we incrementally update the model and iteratively discover new phases from the VR traces. We observe that for DLaaS workloads (in subfigure 2a) the most common behavior (*gray* phase) has high and stable CPU consumption and steady use of memory, while other phases display high variation of Memory (*yellow* phase), CPU/MEM spikes (*blue* phase) or low CPU/MEM usage (*red* phase). In Figure 2b, we show how the default model trained on the DLaaS workload works in the VR workload. We find that the most common behavior is *yellow* phase with extremely high variance of resource usage behaviors, indicating that such a phase may contain unusual behaviors different from what is learned in DLaaS workloads. When updating the *k-means* model by iteratively splitting the *yellow* phase (using the same SSW procedure to find a proper new k for the cluster subdivision), we observe that a new set of phases are discovered for VR workload, and added to the existing model as new centers.

B. Validation of the Phase Prediction

A crucial part of the active auto-scaling pipeline is to predict future resource demands, which we encoded as phases. We tested different models to predict the next phase within a 1 minute time window (4 time-steps), including: 1) The most frequent observed phase as an expected phase for the next time window; 2) the last observed phase as an expected phase for the next time window; 3) an MLP that predicts the next time window; and 4) a LSTM neural network. Table I shows a comparison among the different methods. Note that F1-score quality decreases as we increase the number of steps. We observe that either MLP and LSTM give the best results, but we will use MLP as it is simpler and faster.



(a) Variation in detected phases for DLaaS containers



(b) Variation for VR containers, before and after updating the model

Figure 2: Variation of phases on training and updating

	t=1	t=2	t=3	t=4
Last window mode estimator	0.654	0.614	0.579	0.554
Last observed phase estimator	0.730	0.689	0.648	0.613
One-hot-encoded MLP	0.926	0.876	0.827	0.786
LSTM	0.913	0.863	0.812	0.756

Table I: F1-scores of the prediction models

C. Overall System Performance

Finally, we evaluate the overall system performance under different policies. Here we measure the total slack for the different policies (amount of resource over-provisioning), the number of container kills due to the OOM errors, and the number of resizing actions produced by each policy.

Figure 3 shows the over-provisioning slacks of CPU and Memory for different policies in the Cumulative Density Functions. Note that the phase prediction based proactive policy leads to a tighter resource provisioning with less slack between the limit and the actual usage. Besides, the proactive policy results in fewer container kills (20) due to OOM errors, while both reactive policies result in over 500 container kills. Table II shows the results of auto-scaling performance for each policy in comparison.

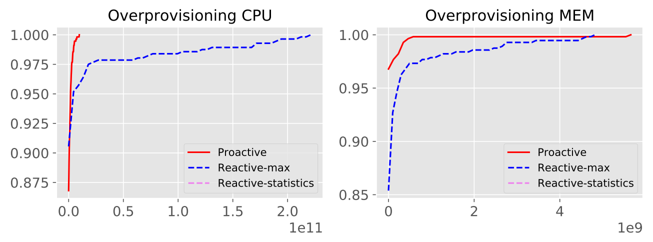


Figure 3: Cumulative distribution of the over-provisioned CPU and Memory for the predictive vs. reactive policies

	Predictive	Reactive-max	Reactive-stats
Auto-scaling changes	3.74	33.28	3.40
OOM containers	20	402	357
CPU Over-provision	6.37e+08	10.20e+08	101.79e+08
Mem Over-provision	5.03e+07	3.74e+07	2.28e+08

Table II: Average number of events for each policy, and over-provisioning in CPU_hour and $KBytes\_hour$

We observe that the average number of auto-scaling changes for the reactive-max strategy is very high, with 33 changes. In contrast, proactive and reactive-statistics policies have a much lower number (around 3-4 changes) per container. Note that both predictive and reactive make minimal changes as the usage fluctuations are probably identified as a particular phase. The proactive policy's principal advantage is to foresee potential phase transitions, avoiding container kills due to OOM errors. Overall, our phase prediction based proactive auto-scaling policy reduces the amount of CPU over-provisioning from $10 \cdot 10^8 CPU \times hour$ to $6 \cdot 10^8 CPU \times hour$. Besides, we find that our proactive policy results in much fewer container kills (20) due to OOM errors, while both reactive policies result in hundreds of container kills, 357 (reactive- statistics), and 402 (reactive-max) OOM errors, respectively. Therefore, our prediction based proactive policy appears to be much safer than reactive approaches with fewer resizing operations.

V. Conclusions

Inspired by prior work [10], [1], we here present a proactive vertical auto-scaling method for containerized machine learning services. We enhance the CRBM based phase discovery method with an iterative, incremental approach to accommodate broader types of machine learning workloads. Besides, via discovering phases for containers running different workload types, we find that typical phase transitions widely occur across containers, contributing to the majority of abrupt resource usage changes in machine learning services. Capable of foreseeing such transitions can help proactively resize containers to accommodate the sudden changes in resource demands. By modeling phase transitions using an MLP model, we prove that such sudden behavior changes can be predicted via phases and can be leveraged in a proactive auto-scaling policy. Evaluations show that our proactive auto-scaling policy can significantly improve resource efficiency more safely (from 65% of potential OOM scenarios to 4%) while maintaining the same auto-scaling amount changes than the best of the reactive policies.

References

[1] J. L. Berral, C. Wang, and A. Youssef, "AI4DL: Mining behaviors of deep learning workloads for resource management," in *12th USENIX Workshop HotCloud*, 2020.

[2] IBM Corp., "Ibm watson machine learning." https://developer.ibm.com/clouddataservices/docs/ibm-watson-machine-learning/. Accessed: 2020-03-15.

[3] S. B. Bishwaranjan Bhattacharjee and et al, "IBM deep learning service," *CoRR*, vol. abs/1709.05871, 2017.

[4] K. R. Jayaram and et al., "Ffdl: A flexible multi-tenant deep learning platform," in *20th International Middleware Conference*, p. 82–95, ACM, 2019.

[5] K. Rzadca and et al, "Autopilot: Workload autoscaling at google," in *EuroSys conference*, EuroSys '20, ACM, 2020.

[6] K. Grygiel and M. Wielgus., "Kubernetes vertical pod autoscaler: design proposal." https://github.com/kubernetes/community/blob/master/contributors/design-proposals/autoscaling/vertical-pod-autoscaler.md. Accessed: 2020-05-29.

[7] S. Baig, W. Iqbal, J. L. Berral, A. Erradi, and D. Carrera, "Adaptive prediction models for data center resources utilization estimation," *IEEE Transactions on Network and Service Management*, vol. 16, pp. 1681–1693, Dec 2019.

[8] S. Baig, W. Iqbal, J. L. Berral, A. Erradi, and D. Carrera, "Adaptive sliding windows for improved estimation of data center resource utilization," *Future Generation Computer Systems*, vol. 104, pp. 212 – 224, 2020.

[9] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *2012 IEEE Network Operations and Management Symposium*, pp. 1287–1294, April 2012.

[10] D. B. Prats, J. L. Berral, and D. Carrera, "Automatic generation of workload profiles using unsupervised learning pipelines," *IEEE Transactions on Network and Service Management*, vol. 15, pp. 142–155, March 2018.

[11] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications' qos," *IEEE Transactions on Cloud Computing*, vol. 3, pp. 449–458, Oct 2015.

[12] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen, "Workload predicting-based automatic scaling in service clouds," in *2013 IEEE Sixth International Conference on Cloud Computing*, pp. 810–815, June 2013.

[13] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *IEEE Intl. Conf. on Cloud Computing*, pp. 500–507, 2011.

[14] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, "Cloudinsight: Utilizing a council of experts to predict future cloud application workloads," in *IEEE Intl. Conf. on Cloud Computing*, pp. 41–48, July 2018.

[15] A. Evangelidis and et al., "Performance modelling and verification of cloud-based auto-scaling policies," *Future Generation Computer Systems*, vol. 87, pp. 629–638, 2018.

[16] A. Ilyushkin and et al., "An experimental performance evaluation of autoscaling policies for complex workflows," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pp. 75–86, 2017.

[17] V. Podolskiy, A. Jindal, and M. Gerndt, "Iaas reactive autoscaling performance challenges," in *IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018.

[18] A. R. Hummaida, N. W. Paton, and R. Sakellariou, "Adaptation in cloud resource configuration: a survey," *Journal of Cloud Computing*, vol. 5, no. 1, p. 7, 2016.

[19] G. Rattihalli, M. Govindaraju, H. Lu, and D. Tiwari, "Exploring potential for non-disruptive vertical auto scaling and resource estimation in kubernetes," in *IEEE Intl. Conf. on Cloud Computing*, 2019.

[20] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *IEEE Intl. Conf. on Cloud Computing*, pp. 500–507, 2011.

[21] G. W. Taylor, G. E. Hinton, S. T. Roweis, P. B. Schölkopf, and T. H. J. C. Platt, "Modeling human motion using binary latent variables," *Advances in Neural Information Processing Systems*, vol. 19, pp. 1345–1352, 2007.

[22] "J-means: a new local search heuristic for minimum sum of squares clustering," *Pattern Recognition*, 2001.