RESEARCH PAPER

An Adaptive Elastic Multi-model Big Data Analysis and Information Extraction System

Qiang Yin · Jianhua Wang · Sheng Du · Jianquan Leng · Jintao Li · Yinhao Hong · Feng Zhang · Yunpeng Chai · Xiao Zhang · Xiaonan Zhao · Mengyu Li · Song Xiao · Wei Lu

Received: 4 June 2022 / Revised: 6 August 2022 / Accepted: 9 September 2022 / Published online: 12 October 2022 © The Author(s) 2022

Abstract

With the diverse applications to industry and domain-specific context, multi-source information extraction on semi-structured and unstructured data, as well as across data models, is becoming more common. However, multi-model information extraction often requires the deployment of multiple data model management, storage, and analysis subsystems on the cloud, many subsystems are not high-resource utilization at the same time, and the resource waste phenomenon is often serious. Therefore, an adaptive scalable multi-model big data analysis and information extraction system is designed and implemented in this paper, which can support data maintenance and cross-model query of relational, graph, document, key and other data models, and can provide efficient cross-model information extraction. On this basis, we can achieve the system resource allocation on demand and fast scaling mechanism, according to the real-time requirements of multi-model big data analysis, and dynamic adjustment of each subsystem resource allocation. Therefore, our solution not only guarantees multi-model query and information extraction performance and quality of service, but also significantly reduces the total consumption of system resources and cost.

Keywords Information extraction · Auto scaling · Multi-model computing · Big data system · Applications to industry/domain-specific context

Feng Zhang
fengzhang@ruc.edu.cn
Qiang Yin
qyin@kingbase.com.cn
Jianhua Wang
jhwang@kingbase.com.cn
Sheng Du
dusheng@kingbase.com.cn
Jianquan Leng
jqleng@kingbase.com.cn
Jintao Li
li.jintao@ruc.edu.cn
Yinhao Hong
hongyh@ruc.edu.cn
Yunpeng Chai
ypchai@ruc.edu.cn
Xiao Zhang
zhangxiao@nwpu.edu.cn
Xiaonan Zhao
zhaoxn@nwpu.edu.cn
Mengyu Li
limy@nwpu.edu.cn
Song Xiao
xiaosongnwpu@nwpu.edu.cn
Wei Lu
lu-wei@ruc.edu.cn
1 Beijing Kingbase Information Technologies Inc., Beijing, China
2 Renmin University of China, Beijing, China
3 Northwestern Polytechnical University, Xi’an, China

© Springer
1 Introduction

In the realm of applications to industry and domain-specific context, information extraction from various semi-structured and unstructured models has become a new study emphasis [1–16]. With the advancement of the big data technology, data now present diversified characteristics. With the guidance of “one size does not fit all” [17], many database systems, data analysis systems, and information extraction systems based on different data models have emerged to solve the problems of data query, analysis, and information extraction in different scenarios. For example, the graph database can be used to store data representing the relationship between data such as social networks and knowledge graph. The document database is used to store complex nested structures such as JSON and XML.

Storing data into databases based on different data models according to their respective characteristics can maximize the information extraction efficiency. However, multi-source information extraction often needs to reference a variety of data, which needs to integrate the data stored in different databases. A traditional way is that the inquirer calls the interfaces of each database by manually writing the query process. This way has high requirements for the inquirer, and most inquirers are unable to complete the query. In recent years, SparkSQL [18], Presto [19], and other new big data analysis systems [20–23] have been developed to solve the problem of data analysis and information extraction in multi-model scenarios, effectively reducing the difficulty of multi-model query.

An adaptive and scalable big data analysis and information extraction system is developed to further reduce the complexity of multi-model query [24]. This system selects the storage subsystem based on data models involved in the analysis process and scales according to the utilization of system resources. By simplifying multi-model query and improving computing power, the analysis system can provide efficient multi-model and multi-source information extraction capability.

In the cloud computing environment, the dynamic expansion and contraction of deployed applications are conducive to the application to deal with the dynamic load. In the face of the rapidly increasing load, scalability can ensure the reliability and availability of application services. On the other hand, the billing mechanism commonly used in the public cloud environment is pay-as-you-go billing. In this mode, the cloud service provider charges according to the actual amount of resources occupied by the application. Adopting the dynamic scaling mechanism can help to reduce the occupation of application resources, thus reducing the deployment cost of the application [25]. Even on the private cloud, the dynamic scaling mechanism can reduce the resource occupation and release the resources to the applications with actual needs, so as to improve the resource utilization and ensure the rational allocation and use of resources.

The analysis system described in this paper is deployed on the cloud platform. With the help of the resource management interface provided by the cloud platform, the adaptive scaling allows the underlying computing engine to expand and shrink dynamically. By adopting the self-learning dynamic threshold scaling strategy based on cybernetics, the system can decrease the amount of resources by 30% on the premise that the impact on performance is less than 5% when evaluated in the cloud workload mode, which greatly improves the cost-effectiveness of resources.

In summary, this work makes the following contributions:

- We conclude the problem of classical threshold methods and propose a self-learning dynamic threshold method based on control theory.
- We design and implement an adaptive elastic multi-model big data analysis and information extraction system.
- We evaluate the proposed method by experiments under real workloads in our system.

2 Background and Related Work

2.1 Multi-model Big Data Analysis

In big data era, data have diversified characteristics. This phenomenon brings multi-model and multi-source big data processing opportunities to the field of information extraction [2, 26, 27]. Many datasets contain data in different formats or data models. Take the Custom-360-view dataset [28] as an example. The dataset provides the researchers with data about customer purchases, including customer social network information suitable for graph storage, commodity information in XML or JSON format, structured customer shopping records, and other unstructured data. Using the database based on graph model, document model, and relational model to store these data, respectively, can make effective use of the characteristics of these data and improve the efficiency of data analysis and information extraction.

However, storing data separately increases the difficulty of data analysis and information extraction. In recent years, in order to meet this challenge, many systems have been developed for multi-model big data analysis and information extraction. SparkSQL and Presto are typical representatives for solving multi-model queries through relational models. These analysis systems take the relational model as the interface to receive SQL queries. By transforming various data...
models into relational models, these systems simplify the complexity of the query model and reduce the query difficulty. However, the computing capability is limited by its single relational model, and it is difficult to query the data model quite differently from the relational model. Such systems that use a single data model query interface are called multistore [29], MISO [30], ESTOCADA [31], etc. make their own optimization on the basis of single interface to improve the query efficiency. HyMAC [32, 33], ReMac [34, 35], and Emacs [36] provide a domain-specific interface to describe big data analysis and execute the computations efficiently on Spark.

In order to solve the problem of multi-model query, another analysis system called polystore [24, 37] puts forward the idea of using multiple data model interfaces to receive queries and using the corresponding data model to perform query calculation. It provides a more flexible choice for multi-model query, but the development of these systems and data source integration are more complex. After data source integration, it may not support the calculation of all data models in the upper layer.

### 2.2 Elastic Scaling

As the amount of data processed in the field of information extraction increases, some work begins to consider putting information extraction in the cloud [38, 39]. This makes resource allocation completely different from performing it locally. Traditional applications run directly in the form of processes, and the operating system can increase or decrease the computing power through the creation and destruction of processes. However, in the era of big data and cloud computing, the multi-process technology of a single operating system can no longer meet the needs. Applications are usually deployed in clusters in the form of container, or disassembled into modules in a microservice architecture [40]. Docker container technology [41], Kubernetes [42], and other container orchestration tools [43] have been widely used. Therefore, a set of resource allocation and management methods are needed to face the changes of the load of the whole cluster, and elastic scaling technology appears.

Elastic scaling technology dynamically adjusts resources according to the change of workload, such as allocating more resources on the same machine (scale-up), or expanding different instances on multiple machines (scale-out). Its main goal is to meet the Service-Level Agreement (SLA) indicators of users and improve the utilization rate of overall resources. When the workload becomes high, in order to prevent the decline of service quality, it is necessary to increase resources in time. Otherwise, we need to recycle resources appropriately when the workload is low to reduce waste.

Profiling application loads is a common method. The results of profiling can be used to help make decisions on resource allocation and scaling. The process of profiling can generally be divided into four stages [44]: (1) Defining data granularity: we determine which indicators need to be monitored and analyzed. We take the cost of data granularity into account because too fine granularity can lead to high overhead, and too coarse granularity can affect the accuracy of analysis results. (2) Data monitoring: the monitoring and collection of data should not affect the monitored application itself, and we need to pay attention to avoid resource competition. (3) Data storage: we need to store the collected indicators and the steps of analysis and processing. Also, it cannot interfere with the profiling. (4) Data processing: we use relevant methods for analysis and modeling. This process may be performed several times to cope with the change of workload at different times.

Paragon [45] and Quasar [46] adopt a collaborative filtering method similar to the film recommendation for profiling. The key technologies are singular value decomposition of matrix and random gradient descent. Quasar carries out parallel fast classification for four categories: scale up, scale out, heterogeneity, and interference. It samples two points for each category and then uses methods of singular value decomposition and random gradient descent to fit the whole curve. If the classification result is inaccurate or the application workload changes greatly, resulting in the decline of quality of service, re-profiling is performed to adjust the resource allocation. Quasar also points out that resource allocation and resource placement should be considered as a whole to reduce interference caused by sharing the same physical resources.

Auto-scaling is another way to automatically scale resources. It aims to dynamically adjust resource allocation in response to workload changes to ensure SLA and reduce overall resource consumption. Auto scaling-related technologies can be divided into the following categories [47]:

1. **Threshold-based method.** This method continuously monitors relevant resource indicators (such as CPU utilization). If the system exceeds or falls below the preset threshold for a certain time, increase or decrease a fixed share of resources (such as 50% CPU), and then, wait for a period of time before next adjustment. The threshold-based method is simple and intuitive. If users have experience in application and set a reasonable threshold, it can make good effectiveness.

2. **Reinforcement learning.** Reinforcement learning uses AI methods to interact with the real environment and learn a model through the feedback rewards, so as to get what kind of resource allocation actions the system should take in different states. The self-adaptive scaling method based on Q-Learning is used to reasonably allocate resources under different VM price models to cope with the change of workload [48]. CDBTune [49] adopts
the method of deep reinforcement learning to automatically adjust hundreds of cloud database parameters. Under six different workloads, CDBTune realizes better adaptability and adjustment effect than other tools and DBA experts. However, reinforcement learning methods usually require dozens of hours of training and cannot cope with the scene of rapidly changing workload.

3. Queuing theory. The queuing theory method models the real-world user request and service processing. Its research content includes three aspects: statistical inference, and systematic behavior, that is, the probability regularity of quantitative indicators related to queuing, and system optimization. However, due to the limitations of mathematics, queuing theory is difficult to shape all the real-world situations.

4. Control theory. It keeps the variables to control at a given level by adjusting the controllable variables. For example, adjust the number of VMs to maintain the CPU utilization of the system. PID [50] is a method with simple structure and implementation, which is widely used in the fields of automatic optimization and mechanical system. More subdivided control theory methods include non-feedback methods, feedback-based methods, and prediction methods, which are often used together with machine learning methods and prediction models.

5. Time series analysis. Many works focus on optimizing time series data [51, 52]. There are also methods of transforming LSTM model and time convolution network into supervised learning problem and using deep learning neural network combined with multiple models for analysis.

In addition, these subsystems are deployed in the cloud environment, and many subsystems cannot have high resource utilization at the same time, which can lead to serious waste of resources. Although the classical threshold method is widely used due to its simplicity and easy implementation, its parameter setting is particularly critical, which directly affects the performance of the algorithm. Usual threshold should be set reasonably based on expert experience or system testing. However, due to its limitation, it cannot perform well in several aspects. When more attention is paid to QoS guarantee, resources can be increased at one time or reserved to deal with sudden load increases, which can lead to a certain degree of waste of resources; on the other hand, if we pay more attention to saving resources and reducing costs, the threshold parameters can be set relatively conservative. When the resources are stretched, the increase in resources can be relatively less, and the recovery of resources can be relatively more. Therefore, the threshold setting of the classical threshold method is an important issue. It is usually difficult to take into account for both QoS guarantee and resource utilization. Improper threshold settings can perform poorly in both aspects.

Therefore, another research motivation of this paper is to use elastic scaling to dynamically adjust resources according to the changes in application load. While ensuring multi-model query performance and service quality, it significantly reduces the overall usage of system resources and costs.

4 System Design

We next show our system design for data analysis and information extraction.

4.1 System Architecture

The overall design goal of the multi-model big data analysis system and information extraction is to establish a cross-model query platform connecting multiple subsystems. The execution ability of the cross-model query is obtained by expanding the multi-model query engine. The bottom layer uses the database supporting various data models to construct the storage subsystems of different models. The system includes query interface layer, computation execution layer, data storage layer, and resource management layer of the adaptive scaling system. The overall structure is shown in Fig. 1.

The adaptive scaling system is based on a large data analysis system oriented to the coexistence of multiple computing modes. The whole system is deployed on the Kubernetes cluster and is managed by Kubernetes for actual resource allocation and container objects. All execution operations of the adaptive scaling require communication with
the management components of Kubernetes. The adaptive scaling system obtains the monitoring index and application status information of the system by Prometheus open source system. The collected system index information is modeled for resource scheduling decisions. The system takes the query performance and error rate of the big data analysis system as the adjustment feedback and adopts a self-learning dynamic threshold algorithm based on cybernetics for resource adjustment. The whole process follows the workflow of index collection, data modeling, and scheduling decision. The adjustment method based on feedback uses the results obtained from the decision for subsequent resource decision-making and improves the resource utilization rate of the system as much as possible on the premise of ensuring the system performance.

4.2 Load Sense

In order to effectively expand the system on demand, it is necessary to monitor the state of the system and related resources. The monitoring data can sense the change of the load and provide it to the expansion strategy algorithm as input data.

Resource utilization indicators, such as container CPU and memory usage, can be obtained through Metrics API exposed by Metrics Server. Metrics Server collects these resource indicators from the Summary API of kubelet public on each node. These indicators can be accessed directly by users, such as using the kubectl top command line, or by controllers in the cluster (such as Horizontal Pod Autoscalers) to make decisions.

In order to automatically collect cluster load, save historical data, and alarm when the load is too high, many resource monitoring tools and alarm tools of open source system are proposed, such as Zabbix [57] and Open-Falcon [58]. The system uses Prometheus [59] as the monitoring alarm module.

Prometheus server finds Prometheus targets to collect resource indicators through Kubernetes exposed services, and the latter exposes resource indicators through specific URLs. The obtained resource indicators exist in the database. The administrator can query the required indicators through PromQL, display resource indicators with data visualization tools, or set alarm rules. When the rules are triggered, Prometheus can notify the administrator through e-mail, chat platform, etc. The information that Exporter, which is used to gather resource metrics, can collect is: kubelet basic running state, container monitoring metrics, host monitoring metrics, pod custom metrics, etc. With the help of Prometheus, we can know the container information of application load and deployment application, and these monitoring information can be used for subsequent resource dynamic scaling decisions.

4.3 Scaling Strategy

In view of the problem of the classical threshold method we mentioned in Sect. 3, we refer to the control theory based on feedback and self-regulation and propose a new self-learning dynamic threshold (SDT) elastic stretching algorithm. Control theory is widely used in automatic management of different information processing systems. The widely used control theory method is based on feedback.
After the decision is made, the system is adjusted according to the actual feedback. There are also some models that use the method of prediction and feedback to modify the prediction model based on the feedback results.

The important difference between the SDT method and the classical threshold method is that the threshold of the stretching resource is not fixed, but dynamically adjustable in the stretching process. At the same time, the adjustment results can be recorded and learned. When similar load scenarios are encountered again, a reasonable initial threshold assignment can be given according to the results of historical learning.

First of all, the adjustable resources are divided into certain granularity. For example, CPU resources can be scaled according to 10% utilization of granularity, and memory resources can be adjusted according to 100M granularity. When there are multiple resources, it can be combined into a slot according to the granularity of each resource, such as 10% CPU, 100M memory as a slot, and the subsequent resource adjustment is scaled in slot.

The utilization rate of CPU, memory, and other resources is continuously monitored during the application load operation. When the utilization rate of resources reaches the alarm threshold, the resources can be scaled. Figure 2 shows the overall flow chart of the expansion strategy.

The expansion of resources can be represented in two stages. The first stage is to calculate the change rate of resources and carry out the initial expansion of resources (such as 50% CPU, 200M memory, or a slot). In the second stage, the feedback-based cybernetics method is used to dynamically adjust the initial resources according to QoS, and the changing rate of resources and the final stable resources adjustment value are recorded in the persistent storage. These two phases are described in detail below, taking resources increases as an example.

When the monitoring alarm occurs for the first time for a certain application load, since there is no relevant historical record, one is randomly selected as the initial resource to add in the range of optional resource increase and then enters the second stage of adjustment. The random initial threshold is likely to be inaccurate, and the increased resources need to be dynamically scaled according to the change of QoS. For example, for the initial increase of 50% CPU, we find that QoS is still not in line with expectations, and then adjust to 100%. If we find that resources are too abundant, we further adjust to 75%. We iterate several times until stabilized. Then, the rate of resources change at alarm time and the final stable resources added value can be recorded to the persistent storage. The last recorded result can be used as the initial resources added value when the similar rate of resources change is encountered. If the initial value is inaccurate due to changes in load characteristics or other reasons, the feedback-based cybernetics process is re-adjusted and the new results are updated to persistent storage, which is also the process of algorithm self-learning.

4.4 Execution Framework

We implement an adaptive elastic scaling strategy on the Kubernetes platform.

Kubernetes is an open-source container scheduling system. Its core goal is to make the deployment, update, and maintenance of container applications more simple and efficient, with the characteristics of portability, scalability, and automation. It originated from Google’s Borg [60] cluster management system. The cluster consists of Master and Node instances, including Scheduler, client management tools, and other components, which provides perfect cluster management and application management functions.

Kubernetes continues Borg’s design concept, using hierarchical architecture. From top to bottom are: (1) Ecosystem: Cluster management scheduling ecosystem. It contains configuration and management systems within clusters such as CNI, Image, Cloud Provider, and external systems including logs, monitoring, and OTS. (2) Interface Layer: Interface layer provides Kubernetes clients and tools. It mainly includes kubelet and kube-proxy running at each working node, and they work together to ensure that user services run in a healthy state. (3) Governance Layer: Management in the cluster is responsible for system measurement, automation, and strategy management. (4) Application Layer: It deploys various applications, and ensures the application layer of

![Fig. 2 Scaling strategy flow chart](image-url)
routing. (5) Nucleus: It provides external API to build high-
level applications, and provides internal core layer of plugin
operating environment.

The management operation unit in Kubernetes is an API
object. Each resource or function in the cluster has its cor-
responding API object to support the management operation
of the resource or function. Among them, pod is the most
basic API object and the smallest unit for running applica-
tions and providing microservices. Users can regard pod as
a host with independent IP, hostname, and process, just host-
ing the service to the host. Kubernetes provides Deployment
and Statefulset resources for application or service upgrade
management, which are, respectively, used to manage and
upgrade API objects with and without state applications.
Users describe the target state that the application or service
needs to achieve in Deployment or Statefulset, and the corre-
sponding controller synchronizes the actual state of the API
object with the expected state without additional operations.

Similarly, users do not need to specify that their applica-
tions or services need to be scheduled on a Node instance,
and this work is done by the scheduler provided by Kuber-
etes. Macroscopically, the scheduler listens to the pod
waiting for new creation through the API server, updates
the definition of pod without node set, and then notifies
the corresponding node to deploy through the API server.
Although the macroscheduling process is clear and easy to
understand, in fact, the scheduler needs to select the best
deployment node through complex scheduling algorithms to
achieve maximum hardware utilization. In Kubernetes, users
can either use the default scheduling algorithm or choose to
use the self-defined scheduling algorithm by creating a new
scheduler object.

When Kubernetes applies for resources, it is only nec-
essary to submit the configuration to the kube-apiserver
component. This component can verify the application
configuration, store the object metadata into etcd [61], and
inform the corresponding controller of the object to generate
a corresponding API object in the cluster. Finally, the kube-
scheduler component can monitor this event, and deploy this
resource on the nodes screened by the scheduling algorithm.

5 System Evaluation

Next, we evaluate our system by the possible load in infor-
mation extraction and data analysis.

5.1 Experimental Setup

We build a Kubernetes cluster and deploy a multi-model big
data analysis and information extraction system on 16 physi-
cal machines with a 24-core CPU, 72GB of RAM and 32TB
of storage each. Based on the methods described above, we
implement an adaptive scaling system, using the threshold
method and cybernetics method for resource regulation.
As shown in Fig. 3, we simulate several different applica-
tion workload patterns. The Cycle/Bursting [62] workload
model is similar to the On-and-Off workload, which is more
suitable for the production environment than the other two
workloads. Therefore, we select the Cycle/Bursting for the
experiment, and the other two workload modes are the same.
We compare the resource utilization and query performance
with and without adaptive scaling enabled under three work-
loads. Moreover, we compare with the threshold method to
evaluate our method’s improvement in resource utilization
and the impact on the system overhead.

5.2 Experimental Analysis

Firstly, we evaluate the improvement in resource utilization
when our method is under Cycle/Bursting workload. Here, a
system without scaling enabled is used as the benchmark for
comparison. Under this condition, we find that enabling the

![Fig. 3 Typical cloud application loads](image)

![Fig. 4 Dynamic query count unit time of the simulated workload](image)
adaptive scaling function can significantly improve system resource utilization.

Secondly, we simulate the Cycle/Bursting load model, and the simulated and actual loads are shown in Figs. 4 and 5. Moreover, we simulate the load situation under three concurrency degrees according to the number of query concurrency. When allocating resources, we allocate CPU and memory resources according to a resource slot, and the allocation of CPU and memory is bound together. The change of resource slot occupation over time is shown in Fig. 6. As load changes, the number of resource slots allocated by the system is dynamically adjusted when the dynamic scaling is enabled. Figure 7 shows the cumulative occupancy of resource slots, which reduces resource occupancy by 74.4%, while the threshold method reduces resource occupancy by 30.4%. Figure 8 shows the average resource utilization of each instance. We can observe that when dynamic scaling is enabled, the utilization of CPU and memory resources is greatly improved. Specifically, the CPU utilization and the memory utilization are increased by 40.7% and 16.9%.

Thirdly, we compare the changes in query performance with and without dynamic scaling enabled. In particular, three queries with a relatively high proportion are selected. As shown in Fig. 9, when the dynamic load is enabled, the query performance change rates are −402.6%, −28.8%, and −33.4%. Meanwhile, the query performance change rates of the threshold method are −35.6%, −54.3%, and −40.4%. Finally, we calculate the change of query error rate with and without dynamic scaling enabled. Specifically, when dynamic scaling is enabled, the correct rate is 97%, which is 3% lower than that of not enabled and within the acceptable range in SLO [63].

Finally, we test our model under stable workload and growing workload [62]. As shown in Fig. 10, under the two workloads, our method reduces resource slot usage by 16.5% and 9.8%, respectively. By contrast, the threshold method reduces by 18.0% and 8.1%, respectively. The above results show that the threshold method is better under the growing workload, and our method is better under the Stable workload.
Overall, we evaluate our system under cloud application workloads. The results show that it can reduce resource usage by 10% to 70% under different workloads while ensuring the user’s query error rate and query performance SLO.

6 Conclusion

The computing engine for multi-model and multi-source information extraction described in this paper is deployed on the cloud platform. With the help of the resource management interface provided by the cloud platform itself, the adaptive scaling allowed the underlying computing engine to scale dynamically. The system was evaluated in the Cycle/Bursting load by adopting dynamic scaling strategies, including threshold method and feedback-based cybernetics. With less than 5% impact on performance and error rates, we reduced resource usage by more than 30%, greatly increasing the cost effectiveness of resources.

Author Contributions Feng Zhang, Yunpeng Chai, and Wei Lu guided this research and proposed the idea. Qiang Yin, Jianhua Wang, Sheng Du, Jianquan Leng, Jintao Li, Xiao Zhang, Xiaoran Zhao, Mengyu Li, Song Xiao, and Yinhao Hong specified and implemented the idea.

Funding National Key Research and Development Program of China (No. 2018YFB1004401).

Availability of data and materials https://gitee.com/fastDataAnalysis/gourd-store.

Declarations

Conflict of interest Not applicable.

Ethics approval and consent to participate Yes.

Consent for publication Yes.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. HL Chieu and HT Ng (2002) A maximum entropy approach to information extraction from semi-structured and free text. In: Rina D, MJ Kearns, and Richard SS (eds), proceedings of the eighteenth national conference on artificial intelligence and fourteenth conference on innovative applications of artificial intelligence, AAAI Press / The MIT Press, Edmonton, Alberta, Canada, p 786–791

2. Dong XL, Hajishirzi H, Lockard C and Shiralkar P (2020) Multimodal information extraction from text, semi-structured, and tabular data on the web. In: Savary A and Zhang Y (eds), Proceedings
of the 58th annual meeting of the association for computational linguistics: tutorial abstracts, ACL 2020, Association for Computational Linguistics, p 23–26
3. Hwang W, Yim J, Park S, Yang S, Seo M (2021) Spatial dependency parsing for semi-structured document information extraction. In: Zong C, Xia F, Li W, Navigli R (eds) Findings of the association for computational linguistics: ACL/IJCNLP 2021, volume ACL/IJCNLP 2021 of findings of ACL. Association for Computational Linguistics, New York, pp 330–343
4. Kim MMH (2017) Incremental knowledge acquisition approach for information extraction on both semi-structured and unstructured text from the open domain web. In Jojo Sze-Meng Wong and Gholamreza Halfar (eds), Proceedings of the australasian language technology association workshop, ALTA, Brisbane, Australia, p 88–96
5. Lockard C, Shiralkar P, and Dong XL (2019) Openencers: When open information extraction meets the semi-structured web. In: Burstein J, Doran C and Solorio T (eds), Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: human language technologies, NAACL-HLT 2019, Association for Computational Linguistics, Minneapolis, p 3047–3056
6. Alonso-Roris VM, Santos-Gago JM, Perez-Rodriguez R, Rivas-Costa C, Gomez-Carballa MA, Anido-Rifon LE (2014) Information extraction in semantic, highly-structured, and semi-structured web sources. Polibits 49:69–75
7. Soderland S (1999) Learning information extraction rules for semi-structured and free text. Mach Learn 34(1–3):233–272
8. Zou M, Yang Q, Jianfeng Q, Li Z, Liu A, Zhao L, Chen Z (2021) Document-level relation extraction with entity enhancement and context refinement. In: Zhang W, Zou L, Maamar Z, Chen L (eds) Web information systems engineering - WISE 2021, lecture notes in computer science, vol 13081. Springer, Cham, pp 347–362
9. Zhang F, Wang B, Li G, Deng L, Xie Y, and Ding Y (2021) TADOC: text analytics directly on compressed graph neural networks via gpu tensor core. In: Proceedings of the 27th ACM SIGPLAN symposium on principles and practice of parallel programming, p 107–119
10. Feng B, Yang T, Li A and Ding Y (2021) Apnn-tc: accelerating arbitrary precision neural networks on amperpe gpu tensor cores. In: Proceedings of the international conference for high performance computing, networking, storage and analysis, p 1–13
11. Feng B, Yang F, and Ding Y (2022) Qgtc: accelerating quantized graph neural networks via gpus on tensor cores. In: Proceedings of the 27th ACM SIGPLAN symposium on principles and practice of parallel programming, p 278–291
12. Wang Y, Feng B, and Ding Y (2021) Tc-gnn: accelerating sparse graph neural network computation via dense tensor core on gpus. arXiv preprint arXiv:2112.02052
13. Feng B, Wang Y, Li G, Xie Y and Ding Y (2021) Palleon: a runtime system for efficient video processing toward dynamic class skews. In 2021 USENIX Annual technical conference (USENIX ATC 21), p 427–441
14. Zhang F, Zheng C, Zhang C, Zhou AC, Zhai J, Du X (2021) An efficient parallel secure machine learning framework on GPUs. IEEE Trans Parallel Distrib Syst 32(9):2262–2276
15. Zhang F, Zhai J, Shen X, Wang D, Chen Z, Mutlu O, Chen W, Xiaoyong D (2021) TADOC: text analytics directly on compressed. VLDB J 30(2):163–188
16. Stonebraker M and Cetintemel U (2005) One size fits all? an idea whose time has come and gone (abstract). In: Aberer K, Franklin MJ, and Nishio S, (eds), Proceedings of the 21st international conference on data engineering, ICDE 2005, IEEE Computer Society, Tokyo, Japan, p 2–11
17. Armbrust M, Xin C, Lian Y, Huai D, Liu J, KBradley, X Meng, T Kafan, MJ Franklin, A Ghodsi, and M Zaharia (2015) Spark SQL: relational data processing in spark. In: Timos KS, Susan BD, and Zachary GI (eds), proceedings of the 2015 ACM SIGMOD international conference on management of data, ACM, Melbourne, Victoria, Australia, p 1383–1394
18. Sethi R, Traverso M, Sundstrom D, Phillips D, Xie W, Sun Y, Yegibasi N, Jin H, Hwang E, Shingte N, Berner C (2019) Presto: SQL on everything. 35th IEEE international conference on data engineering, ICDE 2019. IEEE, Macao, China, pp 1802–1813
19. Zhang F, Zhai J, He B, Zhang S, Chen W (2016) Understanding co-running behaviors on integrated CPU/GPU architectures. IEEE Trans Parallel Distrib Syst 28(3):905–918
20. Zhang F, Zhai J, Shen X, Mutlu O, Xiaoyong D (2022) POClib: a high-performance framework for enabling near orthogonal processing on compression. IEEE Trans Parallel Distrib Syst 33(2):459–475
21. Zhang F, Wan W, Zhang C, Zhai J, Chai Y, Li H, Xiaoyong D (2022) Enabling efficient compressed data direct processing for various databases. In SIGMOD, CompressDB
22. Zhang F, Zhai J, He B, Zhang S, Chen W (2016) Understanding co-running behaviors on integrated CPU/GPU architectures. IEEE Trans Parallel Distrib Syst 33(2):459–475
23. Zhang F, Wan W, Zhang C, Zhai J, Chai Y, Li H, Xiaoyong D (2022) Enabling efficient compressed data direct processing for various databases. In SIGMOD, CompressDB
24. Zhang F, Wan W, Zhang C, Zhai J, Chai Y, Li H, Xiaoyong D (2022) Enabling efficient compressed data direct processing for various databases. In SIGMOD, CompressDB
33. Chen Z, Zhizhen X, Chen X, Soto J, Markl V, Qian W, Zhou A (2021) Hymac: a hybrid matrix computation system. Proc VLDB Endow 14(12):2699–2702
34. Chen Z, Han B, Xu C, Qian W, Zhou A (2022) Redundancy elimination in distributed matrix computation. SIGMOD. ACM, New York, pp 573–586
35. Chen Z, Zhizhen X, Han B, Chen X, Qian W, Zhou A (2022) Remac: a matrix computation system with redundancy elimination. Proc VLDB Endow 15(12):3674–3677
36. Han B, Chen Z, Xu C, Zhou A (2022) Efficient matrix computation for sgd-based algorithms on apache spark. DASFAA. Springer, Cham, pp 309–324
37. Agrawal D, Chawla S, Contreras-Rojas B, Elmagarmid AK, Idris Y, Kaoudi Z, Kruse S, Lucas J, Mansour E, Ouzzani M, Papotti P, Quiané-Ruiz J-A, Tang N, Thirumuruganathan S, Troudi A (2018) RHEEM: enabling cross-platform data processing - may the big data be with you! Proc VLDB Endow 11(11):1414–1427
38. Alexandrescu A (2019) Optimization and security in information retrieval, extraction, processing, and presentation on a cloud platform. Inf 10(6):200
39. Zhang JY, Hu B, He B, Song YB, Zhang GW (2018) Research on online extraction of spatial index information for multi-source surveying and mapping data based on cloud storage. In: Hu S, Ye X, Yang K, Fan H (eds) 26th International conference on geoinformatics, geoinformatics. IEEE, Kunming, China, pp 1–5
40. Tosatto A, Ruiu P, Attanasio A (2015) Container-based orchestration in cloud: state of the art and challenges. Ninth international conference on complex, intelligent, and software intensive systems, CISIS 2015. IEEE Computer Society, Santa Catarina, Brazil, pp 70–75
41. Docker: home (2022). http://www.docker.com
42. Kubernetes (2022). https://kubernetes.io/
43. crio (2022). https://cri-o.io/
44. Weingartner R, Brascher GB, Westphall CB (2015) Cloud resource management: asyure on forecasting and profiling models. J Netw Comput Appl 47:99–106
45. Delimitrou C, Kozyrakis C (2013) Paragon: qos-aware scheduling for heterogeneous datacenters. In: Vivek S, Rastislav B (eds) architectural support for programming languages and operating systems, ASPLOS 2013. ACM, Houston, pp 77–88
46. Delimitrou C and Kozyrakis C (2014) Quasar: resource-efficient and qos-aware cluster management. In: Balasubramonian R, Davis AI and Adve SV (eds), Architectural support for programming languages and operating systems, ACM, Salt Lake City, p 127–144
47. Lorido-Botran T, Miguel-Alonso J, Lozano JA (2014) A review of auto-scaling techniques for elastic applications in cloud environments. J Grid Comput 12(4):559–592
48. Wei Y, Kudenko D, Liu S, Pan L, Wu L, Meng X (2019) A reinforcement learning based auto-scaling approach for saas providers in dynamic cloud environment. Math Probl Eng 2019:1–10
49. Zhang J, Liu Y, Zhou K, Li G, Xiao Z, Cheng B, Xing J, Wang Y, Cheng T, Liu L, Ran M, and Li Z (2019) An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: Boncz PA, Manegold S, Ailamaki A, Deshpande A, and Kraska T, (eds), Proceedings of the 2019 international conference on management of data, SIGMOD conference 2019, ACM, Amsterdam, The Netherlands, p 415–432
50. Borase Rakesh P, Maghade DK, Sondkar SY, Pawar SN (2021) A review of pid control, tuning methods and applications. Int J Dyn Control 9(2):818–827
51. Huang J, Li C, Yu J (2012) Resource prediction based on double exponential smoothing in cloud computing. International conference on consumer electronics, communications and networks (CECNet). IEEE, New York, pp 2056–2060
52. Xie Y, Jin M, Zou Z, Xu G, Feng D, Liu W, Long D (2020) Real-time prediction of docker container resource load based on a hybrid model of arima and triple exponential smoothing. IEEE Trans Cloud Comput. https://doi.org/10.1109/TCC.2020.2989631
53. Chai C, Cao L, Li G, Li J, Luo Y, and Madden S (2020) Human-in-the-loop outlier detection. In: D Maier, R Pottinger, AH Doan, WC Tan, A Alawini, and HQ Ngo, (eds), Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, ACM, Portland, OR, USA, p 19–33
54. Chai C, Li G, Li J, Deng D, and Feng J (2016) Cost-effective crowdsourced entity resolution: a partial-order approach. In: Fatma O, Georgia K, and Sam M, (eds), proceedings of the 2016 international conference on management of data, SIGMOD conference, ACM, San Francisco, CA, USA, p 969–984
55. Chai C, Liu J, Tang N, Li G, Luo Y (2022) Selective data acquisition in the wild for model charging. Proc VLDB Endow 15(7):1466–1478
56. Liu J, Chai C, Luo Y, Lou Y, Feng J and Tang N (2022) Feature augmentation with reinforcement learning. In: ICDE 2021. IEEE
57. Zabbix (2022). https://www.zabbix.com/
58. Open-falcon (2022). http://open-falcon.org
59. Prometheus (2022). https://prometheus.io/
60. Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tun, E, and Wilkes J (2015) Large-scale cluster management at google with borg. In: Reveillere L, Harris T, and Herlihy M (eds) Proceedings of the tenth european conference on computer systems, ACM, Bordeaux, France, p 18:1–18:17
61. etcd (2022). https://etcd.io/
62. Mao M and Humphrey M (2011) Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Lathrop SA, Costa J, and Kramer W, (eds), Conference on high performance computing networking, storage and analysis, SC 2011, ACM, Seattle, WA, USA, 2011, p 49:1–49:12
63. Labidi T, Mtbaa A, Gaaloul W, Tata S and Gargouri F (2017) Cloud SLA modeling and monitoring. In: Liu XF and Bellur U (eds), 2017 IEEE International conference on services computing, SCC 2017, IEEE Computer Society, Honolulu, HI, USA, p 338–345