Edge computing system for large-scale distributed sensing systems

Miloš Simić, Milan Stojkov, Goran Sladić and Branko Milosavljević
Faculty of Technical Sciences, Novi Sad, Serbia
milos.simic@uns.ac.rs

***Abstract*—Cluster orchestration systems achieve high resource utilization, efficient task-packing, process-level isolation. They support high-availability applications with features that minimize fault and recovery time. These systems helped large internet companies like Google, Amazon, and Facebook to satisfy all their applications and big data workloads. With this ability, these systems can run a dozen of tasks at scale, to satisfy all operational needs. But current systems are not that easily applicable in domains with constant data flow with processing on the very edge of the network. We present a system for edge computing, with efficient resource utilization, orchestration, and task-packing. With these properties, a system provides the ability to run user-defined tasks in clusters defined on the edge of the network for constant data acquisition, manipulation, and processing**

Keywords: distributed systems, big data, service-oriented architectures, cloud computing, edge computing, orchestration, containers

I. INTRODUCTION

Nowadays we are facing a massive shift away from the standard, centralized computing model that is provided through cloud computing paradigm.

This shift returns the distribution of computing power back to the edge of the network. Edge network is the idea of connecting sensors to programmable automation controllers (PAC) which handle processing, storage, and communication. The basic concept of edge computing is to leverage new generation technologies, processes, services, and applications that are built to take an advantage of this new infrastructure. The key difference with this model is that it operates and it is deployed on computing hardware closer to the edge of the network. Thus, it is bringing the cloud computing model closer to the ground.

With this new architecture of distributed edge devices, we also need to consider how to manage them and orchestrate jobs to those devices for best resource utilization.

The problem with the current model is that heavily involves centralized architecture. Issues such as latency or small-time delays, security, privacy, network reliability, performance and many others are extremely difficult to completely overcome in centralized computing models. Even a small problem can set a motion to bigger complicated issues [1][2]. For example, Amazon Web Services (AWS) outage is a great illustration of how much large parts of the Internet applications, services, and even businesses depend on the cloud services. If we choose cloud model approach though, then we have two options for data processing: 1) collect all data, store it and then process it - batch processing or 2) use fast data approach, and process data as it is arriving - stream processing. But both batch and stream options imply data must be sent to some cloud provider's cluster. Before processing such data, usually some preprocessing operations (such as filtering, cleaning, etc.) should be executed which involve more time, and from what a cloud provider can economically benefit [3].

Some new problems cannot be fully addressed using just those options. Applications like autonomous driving, smart cities, smart homes or even remote medicine all have different requirements that cannot always be fully addressed by the cloud in its current form. On the other hand, cloud benefits from tools for cluster management and resource orchestration to maximize resource utilization. Such systems should be present in edge computing architecture. Currently, all major cloud providers and companies that are trying to solve those use cases, design their applications and services in a standard, cloud only, way. Some providing more services/features than others, such as storing and data manipulation or insight into data through data analytics, machine learning and data science in general.

This paper is organized as follows. Section II present design of edge computing system. Section III present related work. Section IV summarizes conclusions and briefly propose ideas for future work.

II. DESIGN

The system we propose in this paper is purely on conceptual level. It is influenced by three proven systems used for decades in production environments in large companies for various parts of their systems: 1) Orchestration engine from Google called Borg [4] and it's open source counterpart called Kubernetes [5], that helps Google run their workloads efficiently with maximum resource utilization, 2) Chord [6], a lookup protocol for fast locating the node that store data item and 3) log-structured merge-tree (LSM tree) [7] append only key-value data structure for efficient data storage presented by Patrick O'Neil.

Our system is designed to be able to run tasks defined by users in a similar way those tasks potentially would be run in the cloud. We assume that there are sensors which are scattered over some area and collect data. That data is then forwarded to the distributed nodes which can store data and run user-defined tasks. Since this part of the system runs those tasks on data that is coming from some

sensing source on the edge, we ensure that data sent to the cloud for future analyses is smaller and already pre-processed in some degree. In this way, the system enables faster time to market approach saving both, time and money to users.

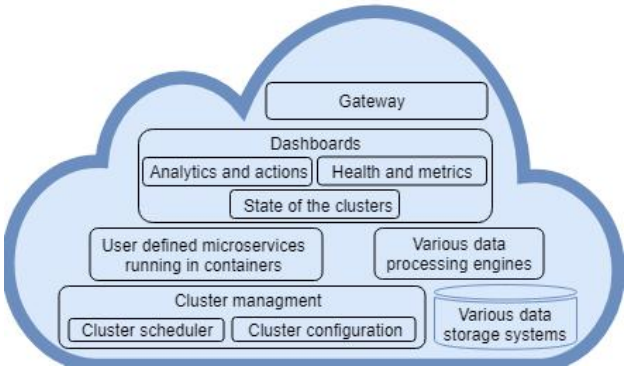
In order to gain maximum resource utilization from these distributed nodes, we define a master process which resides in the cloud and manipulates the information about previous, active and future user tasks which will be triggered.

The system we propose is separated into two major parts: 1) master process that is responsible for orchestration and task scheduling, storing information about user-specified tasks, cluster topology and resource utilization, 2) cluster of nodes, that is responsible for storing data and running user-defined tasks.

1. *Cloud master process*

In the cloud, we would have logic that orchestrates user-defined jobs, store and process metrics, keep secrets and run user-defined microservices. Also, have engines to process data collected from clusters.

Picture 1 shows high architecture overview of the cloud system.



Picture 1. High overview of the cloud architecture

The job is a standard user-defined application, which can be categorized into one of three different types: 1) batch jobs, for standard batch processing over some collection of data. This job should run only when data is available, or at predefined time 2) events, this type of job should react when some specific event happened in the system, or if acquired data pass some predefined threshold 3) streaming jobs, should continually do some processing on data as it is arriving (long-running jobs). Every job should contain direct acyclic graph (DAG) [8] which is created when job transits from operation to operation and result of every operation is stored to disk or memory depending on user's preference. This is done to prevent data loss if the job fails or entire node fails or restart. If fail happens, the job can continue from last saved checkpoint. The orchestration process is composed of a key-value store (optimized for scaling) and job scheduler. Key-Value store [9][10] contains topology and information of the edge, user-defined job definitions, number of replicas for every job, the total amount of computing resources such as CPU, RAM, disk storage in every edge cluster, configuration for every specified job.

A job scheduler is responsible to find node, schedule jobs there and always keep running a defined number of jobs. Key-Value store is replicated in an odd number of times. To prevent potential chaos, the consensus protocol is responsible for maintaining global truth. Beside orchestration process, in the cloud users can register their own applications that will contain data from jobs they submitted, run some analytics for data insights and monitor their portion of the system with dashboard and monitoring tools. Every part in the cloud is running as a single service (microservice).

With this approach, we can more easily scale, and keep the system always available [11][12]. All services should be available to outside world through dashboards and/or Representational State Transfer (REST) services, and inside communications should be binary for less overhead. Cloud should provide dashboards so that users can easily monitor various parts of the system.

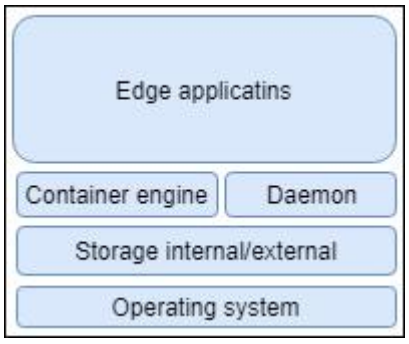
2. *Edge cluster*

The second part is edge cluster. The core concept here is a node. Node is a computer that runs an operating system, container engine, and a small daemon. Daemon does all the communication with orchestration engine in the cloud. He is also responsible for metrics collection, and communication between nodes inside the cluster, starting/stopping/restarting jobs. Since this tasks should run in parallel, we propose using Actor model [13] where daemon just pass the message to some of its children tasks.

Nodes are connected in the cluster in peer to peer [14] manner in order to eliminate a single point of failure and to be easier to scale. All nodes are equal, there is no some special node like the master node. Nodes that belong to the same cluster form a distributed hash table (DHT)[6] used for data storing, lookups, deletes but also run user-defined jobs scheduled in orchestration engine in the cloud.

All data stored in DHT is replicated by a user-defined number of times in subsequent nodes to prevent data loss. Nodes and data are hashed using consistent hashing principles and all nodes that belong to the single cluster are connected in one virtual ring. With this simple idea, we can easily partition data so each node is only responsible for a small portion of data. Consistent hashing [15][16] is beneficial for two reasons: 1) if a node join/leave a group, we just need to copy a small amount of data over network because each node is only responsible for small partition of data, 2) we store similar data (same key) on the same place (same node) when we do some calculation, since it is much faster because data is already on the same node. By doing this we benefit because we bring calculation to the data. Each node has a retention policy, or how long will they keep the data. Because these nodes are not high-end machines with a lot of resources.

Picture 2 shows a high overview of a single edge node with running processes.



Picture 2. High overview of single edge cluster node

The system must always be able to store new data. Each node can discover other nodes that belong to the same cluster using Gossip-based protocol [17]. Each node in the ring knows only about a handful of nodes clockwise from it. This is done in order to scale the whole system. Each node can contact any other node in the cluster for some data, and eventually will get information where that data is actually stored. All communication between nodes should be as fast as it can, so binary protocols like HTTP/2 [18] should be used via remote procure call mechanism (RPC) [19][20]. Every job that runs on these nodes, should be packed in form of a Linux container [21][22]. This gives us the better ability for packing jobs on machines, easier orchestration and separation between jobs on a single machine.

The benefit of this approach is that users are free to define their own applications that are composed of multiple different jobs, and can always change, upgrade/downgrade or remove their applications depending on workloads they need.

III. Related work

There are attempts both in academia and industry that are trying to combine cloud with the edge, or to go even further.

For example, Thinnect [23] is an IoT edge network service provider which combines cloud and edge computing. Their service is complementary to cellular operators, offering high density, large scale, local coverage, utilizing cost-efficient technologies. Since their approach is more micro-controller based, for specific tasks we need specific devices.

Nebula [24] is the project which goal is to be a Docker orchestration for IoT devices and distributed services (like CDN or edge computing). Users have the ability to simultaneously update tens of thousands of IoT devices all around the globe with a single API call, allowing users to treat IoT devices like another distributed Docker application. The project is at an early stage, but despite all of that, as an idea, it provides a lot of potentials.

Mainflux [25] is highly secure, scalable, open-source IoT platform written in Go and deployed in Docker. It serves as software infrastructure and set of microservices for development of the Internet of Things Solutions and deployment of Intelligent products.

IV. Conclusion

Since we are facing massive shift from centralized computing architectures, we should find a new way to address problems with new applications like the internet of things area and autonomous driving. Cloud computing is a great tool used for many decades now, but its centralized nature may not be well suited for this new kind of applications. Since we are waste a lot of time and money on storing filtering and preprocessing data in the cloud we could consider the option to do preprocessing where data is created and that is on the very edge of the network. Thus edge computing model could be used to help cloud computing paradigm and users to preprocess data on the edge of the network, and only that data send to further analysis and on that way accelerate time to market for its users.

Security is a crucial part of every system, and this paper does not cover this topic of overall system design. The future work will focus more on the security of the whole system, beyond just process insolation trough containers. Also, we will focus on metrics collection and keep the health of the entire system and every job interdependently, and self-healing trough task orchestration. Another important topic is configuration management of every job. Since we are talking about large-scale distributed sensing systems manual configuration of every node is out of the question. We must find a way to automate the whole process.

References

[1] H. Gunawi, M. Hao, R. Laksono, A. Satria, J. Adityatama, K. Eliazar, Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages, SoCC '16 Proceedings of the Seventh ACM Symposium on Cloud Computing Pages 1-16

[2] J. Hamilton, On Designing and Deploying Internet-Scale Services, Pp. 231-242 of the Proceedings of the 21st Large Installation System Administration Conference (LISA '07) (Dallas, TX: USENIX Association, November 11-16, 2007).

[3] R. Perry S. Hendrick. The Business Value of Amazon Web Services Accelerates Over Time, https://media.amazonwebservices.com/IDC_Business_Value_of_AWS_Accelerates_Over_time.pdf, last accessed 21 april 2018

[4] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at Google with Borg, Proceedings of the European Conference on Computer Systems (EuroSys), ACM, Bordeaux, France (2015).

[5] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, Omega, and Kubernetes, ACM Queue, vol. 14 (2016), pp. 70-93.

[6] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for Internet applications, IEEE/ACM Transactions on Networking (Volume: 11, Issue: 1, Feb 2003).

[7] O'Neil, Patrick E.; Cheng, Edward; Gawlick, Dieter; O'Neil, Elizabeth (June 1996). The log-structured merge-tree (LSM-tree). Acta Informatica. 33 (4): 351–385. doi:10.1007/s002360050048. Retrieved 2014-08-03.

[8] M. Fiore, M. Devesas Campos, The Algebra of Directed Acyclic Graphs, Computation, Logic, Games, and Quantum Foundations, 2013.

[9] M. Berezecki E. Frachtenberg M. Paleczny K. Steele, Many-core key-value store, IGCC '11 Proceedings of the 2011 International Green Computing Conference and Workshops Pages 1-8.

[10] A. Nayak, A. Poriya, D. Poojary, Type of NOSQL Databases and its Comparison with Relational Databases, International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868

Foundation of Computer Science FCS, New York, USA Volume 5– No.4, March 2013.

[11] N. Dragoni, I. Lanese, S, Larsen, M. Mazzara, R. Mustafin, et al.. Microservices: How To Make Your Application Scale. A.P. Ershov Informatics Conference (the PSI Conference Series, 11th edition), Jun 2017, Moscow, Russia.

[12] W. Hasselbring, Microservices for Scalability: Keynote Talk Abstract, ICPE '16 Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering Pages 133-134.

[13] C. Hewitt, Actor Model of Computation: Scalable Robust Information Systems, eprint arXiv:1008.1459 2010.

[14] M. Bawa , B. Cooper , A. Crespo , N, Daswani , P. Ganesan , H. Garcia-Molina , S. Kamvar , S. Marti , M. Schlosser , Q. Sun , P. Vinograd , B. Yang, Peer-to-peer research at Stanford, ACM SIGMOD Record, v.32 n.3, September 2003

[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. SIGOPS Operating Systems Review, 41(6):205–220, 2007

[16] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing (STOC), pages 654–663, 1997.

[17] A. Das, I. Gupta, A. Motivala, SWIM: scalable weakly-consistent infection-style process group membership protocol, Proceedings International Conference on Dependable Systems and Networks

[18] S. Ludin, J. Garza, Learning HTTP/2 A Practical Guide for Beginners, O'Reilly Media 2017, ISBN-10: 1491962445.

[19] S. Wilbur, B. Bacarisse, Building distributed systems with remote procedure call, Software Engineering Journal (Volume: 2, Issue: 5, September 1987)

[20] A. Birrell, B. Nelson, Implementing remote procedure calls, ACM Transactions on Computer Systems (TOCS) TOCS Homepage archive Volume 2 Issue 1, February 1984 Pages 39-59

[21] A. Javed, Linux Containers: An Emerging Cloud Technology.

[22] D. Bernstein, Containers and Cloud: From LXC to Docker to Kubernetes, IEEE Cloud Computing (Volume: 1, Issue: 3, Sept. 2014)

[23] Thinnect, http://www.thinnect.com/, accessed 21 April 2018

[24] Nebula, https://nebula-orchestrator.github.io/, accessed April 2018

[25] Mainflux, https://www.mainflux.com/, accessed April 2018