Astro - A Distributed Discovery and Lookup Service for Edge networks

Meghana N Babu
mbabu7@gatech.edu

Mugdha Bondre
mbondre3@gatech.edu

I. INTRODUCTION

Cloud computing has revolutionized the availability and accessibility of resources. With increasing amounts of data being generated at the edge of the network, it is impossible to send all the data to the cloud for pre-processing without compromising on latency and clogging the network bandwidth. Network latency can have serious impacts in the operation of edge devices which require real-time data processing, such as autonomous vehicles.

Fog computing came into being to tackle this problem by extending cloud computing to the edge of the network [1]. This new paradigm is supported by the computing power provided by the vast number of mobile edge devices present today. This has opened a new array of services and applications that are at a close proximity to the end-users.

A distinguishing characteristic of fog computing is its dense geographical distribution of heterogeneous nodes such as Android devices, resource-limited structures like Raspberry Pi, or embedded devices like network-connected sensors, and routers. If we can harness the data and compute resources of these nodes, we can develop a large-scale service infrastructure which can be scaled according to the requirements of the application.

The heterogeneous nature of the system poses a lot of challenges in resource management of the underlying infrastructure [2], [3]. The foremost challenge is the discovery of such resources. The system can be visualized to comprise of two main parts, first - device owners who are willing to lease their resources, and second - users who want to use these resources. Our project aims to develop a service, *Astro*, for resource discovery, registry, sharing and access management in such an infrastructure. Astro is built on top of ZooKeeper, a light-weight coordination system. We present the underlying three-layer architecture of Astro. Taking into account the high mobility in the edge network and the recent release of ZooKeeper (ZK) to support dynamic reconfiguration of ZK servers [4], we present the implementation of the first layer using the ZK service [5].

II. RELATED WORK

Platform architecture Seattle [6] is a practical and publicly accessible fog computing platform which has a unique componentized architecture to enable real-world application deployment on heterogeneous nodes. The architecture incorporates a lookup service which is a generic network-accessible key/value store that the resource manager and experiment manager can use to announce and retrieve information about sandboxes. The paper mainly focuses on providing resources on heterogeneous nodes, but does not give much detail on how the resource discovery and access management are accomplished.

Resource lookup Datta et al.[7] define a centralized search engine for resource discovery of IoT devices. The problem with this approach is that only a mediator node has the knowledge of all the resources in the network - this is not practically scalable and does not predict execution time efficiently for appropriate reallocation of resources. Rodrigues et al. [8] proposed a single hop distributed hash table, and Douceur et al. [9] present a system that routes in a constant number of hops. Both the systems assume very small peer dynamics which is generally not the case with the networks in reality. The issue of scale was addressed by FastTrack [10] and Gnutella [9], [11], [12] in unstructured P2P applications, and by Mizrak et al. [13] in structured P2P applications where they introduced Superpeers by adding an additional hop in the lookup algorithm in a large-scale network. Although this might prove to be efficient (constant time as claimed by the authors), this is not directly applicable to an edge network.

Locality based lookup A. Salem et al. propose Kinaara, an edge computing framework offering discovery capabilities [14]. It is a multi-tier architecture to organize geographically proximal edge devices with the help of trusted special mediator nodes, that keep a connection to other such mediator nodes in the network. Kinaara also suggests a novel encoding method for resource identification. It derives its indexing scheme from Chord [15], and improves it by basing the index on the similarity of the proximal resources to efficiently allocate resources upon request. Some more work towards the resource management in fog computing involves techniques such as game theory [16], [17], cooperative management [18], multiple tiers involving the benefits of a cloud-tier [19], and linear programming [20]. Although Kinaara [14] describes an efficient architecture for multi-tier resource discovery and allocation, many of the implementation details are foggy.

(a) The design defines the mediator as nothing more than a gateway node connecting its proximal ring to other mediators. However, the implementation hints that a mediator alone stores information about users and resources allocated to them in its respective proximal ring, making it a single point of failure. If a mediator goes down, the allocation information is lost, thus exposing a loophole in Mobility Management of Kinaara.

(b) The mediator node is responsible for both device join and resource allocation. This would lead to a lot of overhead in the mediator node to handle multiple concurrent requests, considering the high mobility in the edge network in real-life.

Access management Seattle [6] provides a trust management intermediary in the form of clearinghouse, which is a human interface to request and deliver sandboxes. Although there has been significant research in the area of security and privacy for fog computing [21]–[27] literature on access management for resource sharing is rather scarce. Fog computing involves different kinds of resources - compute, storage, network and security. Read-only storage resources can be allowed to be shared among different users at the same time given that the device owner of the resource approves its access rights.

III. ARCHITECTURE

Our project derives a holistic idea of the resource sharing and defragmentation system from Seattle [6], and focuses on developing a unique lookup directory service for the architecture. The three-layer architecture presented in Kinaara [14] is taken as an inspiration for the architecture of Astro as seen in Figure 1. Starting from the bottom is the *edge device* (presented in various shapes) that is willing to lend it resources to be used by the users of Astro. Each edge device belongs to a *proximal cluster*. The proximal cluster is formed taking into account the locality of the devices and their network provider. Each proximal cluster has a leader called the *mediator*, which helps in communications across proximal clusters. The mediators from each proximal cluster form the *mediator cluster layer*. For the purposes of this project, we assume a *cloud service* that is used as the primary point of contact for any edge device that wants to join the Astro system. The primary functionality of the cloud service is to inform the new device of the closest proximal cluster that it can join.

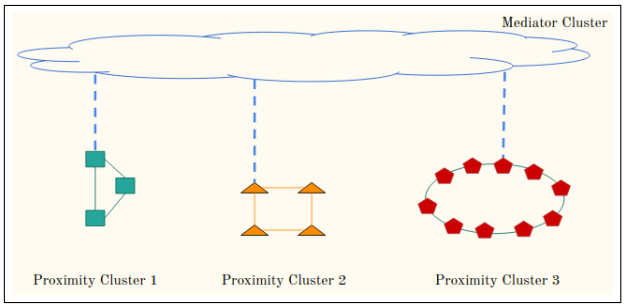

Fig. 1. Astro Architecture

IV. IMPLEMENTATION

Astro is still in its nascent stages and this work presents the implementation details at the proximal cluster layer. We define proximal cluster in our implementation at the WiFi Access Point (AP) level that encompasses a single wireless LAN where devices communicate using wireless AP(s). Mobility is a common challenge to be handled in edge network, where multiple edge devices join and leave the proximal cluster frequently. We propose leveraging the novel dynamic reconfiguration in ZK to handle such mobility [29]. The next section is dedicated to briefly explain ZK, followed by the section that describes Astro's implementation and functionality.

A. ZooKeeper

ZooKeeper (ZK) is a service for coordinating processes of distributed applications. It incorporates elements from group messaging, shared registers, and distributed lock services in a replicated, centralized service [5]. It implements *a coordination kernel* which manipulates simple wait-free data objects organized hierarchically. This enables developers build more complex coordination primitives. Astro leverages various capabilities of ZK:

- *Group Membership*: The proximal cluster of Astro is implemented as a ZK Ensemble. This guarantees replication on each of the nodes in the cluster and read consistency.
- *Leader Election*: Each proximal cluster requires to have a mediator which can connect the cluster to the mediator cluster and facilitates new device join. Thus, the leader functionality provided by ZK is useful for the election of a mediator of a cluster.
- *ZNode Structure*: ZK provides a hierarchical structure of ZNodes which is a versatile feature useful for developing complex functionality. Astro employs this feature to record availability status of resources. The system also uses the ZNode structure for executing two-phase commit for resource allocation.
- *Dynamic Configuration Management*: Starting from version 3.5.4-beta, ZK comes with full support for automated configuration changes: the set of ZK servers, their roles (participant / observer), all ports, and even the quorum system can be changed dynamically, without service interruption and while maintaining data consistency [28]. This is a very beneficial development for edge computing given the high mobility of Edge devices. Astro assumes that devices will be leaving and joining different proximal clusters frequently and hence leverages the dynamic reconfiguration of ZK. This helps maintain a consistent state of devices and resources in a cluster.

For dynamic reconfiguration, the administrator must make sure that a quorum (majority) of participants from the new configuration are already connected and synced with the current leader. Without a valid quorum, the cluster crashes. The failure of a single follower does not prevent servers from forming a quorum. However, leader re-election following failure of the leader can take up to 200ms. From here on, every reference to ZK means that we use ZK with dynamic reconfiguration, version 3.5.4-beta.

B. Astro

The implementation of Astro currently at the proximal cluster layer is based on ZK. Each proximity cluster starts at the AP as it comes up with three instances of ZK. This optimization ensures availability of the proximal cluster because of the following reasons –

- A WiFi AP is a stable element in the edge network. Starting three instances on such stable device per ensemble ensures that there is always a quorum up and running for any new device that wants to join.
- When an AP starts three ZK instances, a leader is chosen from one of the three instances. This ensures that the leader is present in a stable entity of the ensemble reducing the frequency of the leader election downtime.

Resource Encoder: The implementation of Astro contains a rudimentary resource encoder. The metadata of every resource that a device wants to add, including the resource type, capacity guarantees, and read-only flag are fed into this encoder. The encoder returns the encoded resource object with its *allocated flag* set to *false*. The encoder is guaranteed to uniquely encode every resource object.

Current State of the system: Let us consider a WiFi AP, AP_1, that belongs to the network provider NP_1. We assume that the new edge devices that want to join the proximal cluster are in the locality bounds of AP_1. For the purposes of the discussion of functionalities of Astro in the further section, we consider the current state of the proximal cluster as shown in the Figure 2. AP_1 is running instances of ZK 1, 2 and 3. The cluster also contains edge devices D_4, and D_5, each running one instance of ZK. The orange node corresponds to Device D_6 that belongs to NP_1, which wants to join the cluster.

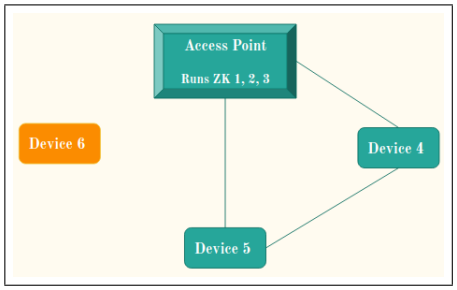

Fig. 2. Initial state of Proximity Cluster

The proximal cluster in Astro stores the resource information (type, metadata, owner, allocation state, etc.) using the ZNode structure of ZK. Figure 3 shows the current state of the resources in the proximal cluster of AP_1. The resources are broadly classified into *compute*, *network* and *storage* resources as shown. Each resource R_i under these resource type ZNodes corresponds to an encoded resource of one of the devices D_4 and D_5.

Device Join: The following are the steps taken by D_6 when it wants to join the proximal cluster hosted by AP_1:

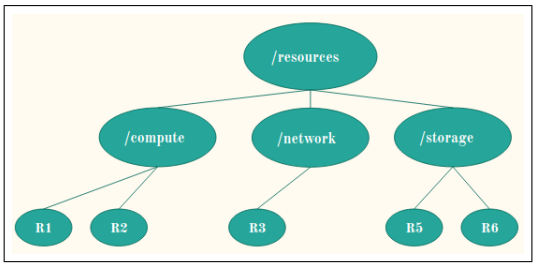

Fig. 3. Initial state of ZNodes

I *Prep Start ZK*:
- Contacts cloud service to get the information about the proximal cluster closest to it and hosted by NP_1. Cloud service responds to D_1 with the contact of AP_1.
- Contacts AP_1 to get the current committed dynamic configuration of the ensemble.
- Adds itself to the local copy of the received config.

II *Start ZK*: Starts an instance of ZK with the updated local copy of the config from step I.

III *Join Ensemble*: D_6 joins the ensemble of AP_1 by updating the config of the ensemble.

Add Resource: When a device, say D_6, wants to add its resource into Astro to lend it to the users, the following steps are taken:

I *Encode Resource*: D_6 sends the metadata about the resource – address and port, capacity guarantees, read-only flag – to the resource encoder and get the encoded resource, say R_4.

II *Create ZNode*: The resource type RT (one of *compute*, *network*, or *storage*) of the resource to be added is selected and the ZNode for the resource is created as $/RT/R_4$.

User Request: When a user U_1 contacts the AP/a device of a proximity cluster, it sends the metadata about the resource it wants to lease. The metadata information is very similar to the metadata information sent by the device when it adds a resource. The following steps are taken in order to lease a resource to U_1:

I *Get Resource Candidates*: Based on the resource type of the request, the resource type ZNode is chosen. Then, Astro checks for the metadata information passed and provides U_1 with the list of possible resources in the order of preference that U_1 can try to allocate. There is no added overhead on the AP itself to serve all user requests, thereby offloading the allocation work to the user itself.

II *Resource Connect*: Once U_1 receives the resource candidates, it tries to see if it can allocate one of the resources to itself in the order provided to it. Astro uses a novel two-phase commit algorithm to handle concurrent users trying to allocate the same resource to themselves, which is described below.

Phase I – *Claim Ownership:* A user is defined as the *owner* of the resource if the data of the resource ZNode contains that user's ID. To check if the resource candidate RC is still available to be allocated, U_1 first checks if the *allocated flag* of RC is set to *false* and the data of the RC ZNode is empty indicating that there is no other owner of this resource. Once these two checks pass, U_1 sets itself as the owner of the RC.
Phase II – *Allocate Resource:* ZK allows multiple writes to be written on the same ZNode. If concurrent users enter the Phase II of the commit algorithm, they check to see if they are still the owner of the RC. Only one of the users pass this check, say U_1, which goes and sets the data of the RC to its user ID and changes the *allocated flag* to *true*.

III *User Leave*: When U_1 is done using the resource, it deallocates the resource from itself by releasing the ownership and setting the *allocated flag* to *false*.

Device Leave: When the device D_6 decides to leave, it needs to perform a series of steps to ensure a graceful exit.

I *Remove resources*: All the resources owned by D_6 need to be cleaned up. One or more of the resources of D_6 can be allocated to users when D_6 decides to leave. In such a scenario, D_6 can decide whether to forcefully exit or wait for the users to complete their operation before exiting through the *force_exit* flag.

II *Leave ensemble*: The dynamic config of the ensemble is updated to remove the participation of D_6 in the ensemble. In such a case, since ZK is still running on D_6, it is considered as a non-voting member of the ensemble.

III *Stop ZK*: Finally, the ZK server running on D_6 is stopped.

Handling crashes: Crashes are an inherent part of edge computing. Astro's architecture dedicatedly handles device crashes by means of a Garbage Collector. Garbage Collector is a background thread having different operations depending upon whether the device running it is a leader node or a follower node in the ZK ensemble. Garbage Collector (GC) takes up one of the two following roles:

The functionality of the Garbage Collectors (GC) on different nodes is as follows:

1) *Leader GC*: The GC running at the leader node handles the crashes happening in a cluster. The GC runs periodically and checks if all the ZK servers registered in the config of the ZK ensemble are up and running. This is carried out using a ZK monitoring command - "ruok" which is run periodically by the GC. Live ZK servers respond to the command as "imok". GC removes all the servers which do not respond to the monitoring command and removes all the resources offered by the crashed server/device. As GC runs a background thread, it does not affect the latency of Astro as a service.
2) *Follower GC*: The main functionality of the GC running at the follower nodes is to handle device mobility. If a device (which internally runs a ZK server) is moving, it should be seemlessly able to leave a proximal cluster and join another based on its location or proximity. The GC periodically checks if the node is a part of any ZK ensemble, if not, it contacts the cloud service to find the nearest AP and performs a device join in that cluster. Follower GC is currently under development.

V. EXPERIMENTATION AND ANALYSIS

In this section, we evaluate Astro through a local setup of a proximal cluster. The cloud service is assumed to exist and the system starts with three instances of ZK in an ensemble, which serve as the AP. The contact of the AP is assumed to be "well-known". We measure the latency of each of the above functionalities explained in section IV-B. We explain the various experiments run, the findings from them and our analysis of each of them in this section.

Device Lifetime Latency: The first batch of experiments aims to measure the latency of various phases in the lifetime of a device. Here, the lifetime of a device consists of four phases –

I Device Join
II Add Resources
III Lease Resources
IV Device Leave

We have measured the latencies for Phases I, II and IV for single as well as concurrent requests from 3 servers for 20 runs. Figure 4 shows the results for multiple devices in the three phases acting concurrently. As it can be observed, out of the three phases, Phase I has the highest latency.

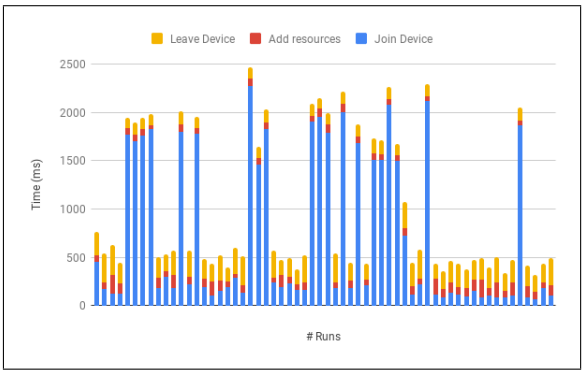

Fig. 4. Latency of different phases of a device's lifetime over 20 runs

To analyze it further, we measured the latency of the 3 subphases of the Device Join Phase as defined in Section IV-B. Distribution of the three subphases in the average case can be seen in Figure 5. Prep Start ZK takes 12.4% of the total time, Start ZK phase takes only 1% of the total time, whereas Join Ensemble dominates the latency of Device Join by consuming 86.6% of the total Device Join time. This behavior is expected in the case of concurrent device join requests as devices contend change the ZK ensemble config

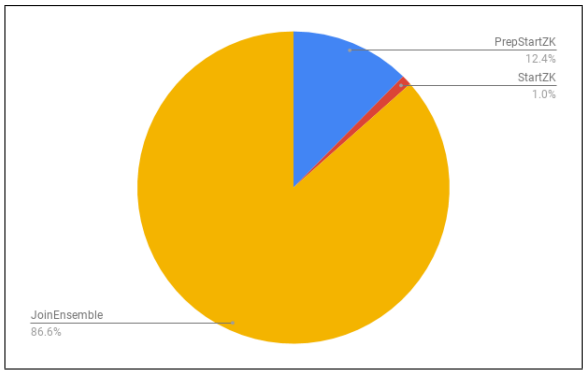

Fig. 5. Latency distribution over different phases of a device join operation

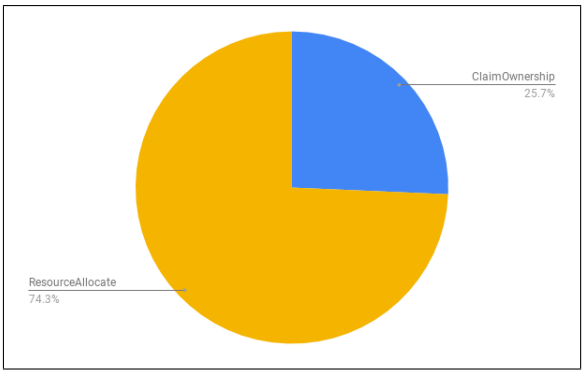

Fig. 7. Latency distribution over the two phases of Resource Connect

to add themselves to the ensemble. In case of concurrent requests, ZK allows one request to proceed, while others encounter an Exception and back off with a randomized timeout between 0ms to 100ms. Because of the concurrent requests, we see a variation in Join Ensemble from 36ms to 2056ms. In case of single requests, Join Ensemble phase varies from 37ms to 96ms, which proves the variation to be influenced by concurrency.

User Request Latency: The next batch of experiments focus on the latency of the user requests. The phases of the user request are as explained in section IV-B. The experiments were run for single as well as concurrent user requests. All user requests were configured such that they are all successfully satisfied by the underlying resources in Astro. The concurrent requests were configured such that every request had more than one resource candidate, and that there was contention on the resource ZNodes.

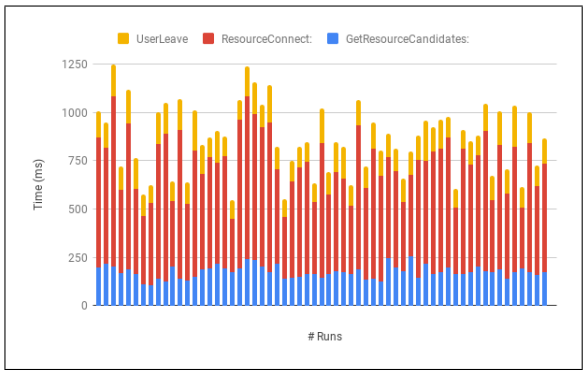

Fig. 6. Latency of different phases of a user request over 20 runs

Figure 6 shows the latency results of different phases of three concurrent user requests over 20 runs. It is clear from the figure that the second phase, Resource Connect, has the highest latency. The latency of the first phase, Get Resource Candidates, is unavoidable because the resource candidates algorithm is pessimistic. It provides all possible resources for the resource requested which makes the complexity of this phase linear in the size of number of resources of the requested resource type in Astro. The latency of the third phase, User Leave, is unavoidable since the user needs to perform the safe operations of releasing the ownership and deallocating the resource from itself.

Figure 7 shows the latency distribution over the two phases of the two-phase commit algorithm. 25.7% of the time is taken by Phase I, Claim Ownership, and 74.3% of the time is taken by Phase II, Resource Allocate. We can attribute this finding to the fact that the steps that need to be taken in the Resource Allocate phase are communication intensive. The Resource Allocate phase is unavoidable. This made us explore the Claim Ownership further to come up with an optimized approach that eliminates this phase altogether making this a one-phase commit algorithm.

VI. NOVELTY

Astro stands out from the existing edge services in multiple ways:

1) Astro offers the novel idea of leveraging ZK to edge network, which has not been explored before. We attribute this observation to the fact that until recently, ZK only offered static configuration files, and the addition of supporting dynamic configuration files from version 3.5.3 onwards can be used to handle the high mobility in edge networks [4], [28].
2) One of the main contributions of Astro is that it allows read-only resources to be shared with multiple users in the edge network.
3) Astro reduces the load on the mediator node by offloading the burden of resource allocation on the user itself.
4) Astro ensures that the resource allocation is safe for concurrent users contending on the same resource through a novel 2-phase commit algorithm.
5) Astro is also tolerant to device crashes. The efficient garbage collector handles crashes gracefully to ensure a clean directory.
6) The two main drawbacks of ZK are – (1) removing the leader will result in a short unavailability, (2) it expects a quorum to be present every time reconfiguration is run [29]. Astro mitigates the frequency of these two events from occurring by ensuring that a quorum is maintained in the stable WiFi Access Point and that the ZK leader is elected from one of the three instances of ZK running on the AP.

VII. LIMITATIONS

Since ZK dynamic reconfiguration is fairly new and still in its beta version [28], it comes with a limitation that it can support only up to 256 nodes in an ensemble. In Astro, one ZK ensemble is used per proximal cluster, which would not suffice for the number of nodes joining a single WiFi AP in real-life [30].

VIII. FUTURE DIRECTIONS

Astro is a budding project with various opportunities for further enhancements.

1) *Resource Connect algorithm*: Two-phase commit adds a latency overhead of 65% during the user action of resource lookup. The *Claim Ownership phase* adds a lot of communication overhead when there are multiple concurrent user requests on the same resource since ZK consults all nodes before committing the data on the ZNode. This communication overhead can be reduced by optimizing the resource lookup to a single phase through efficient exception handling.
2) *Mediator cluster*: Astro currently has the support for a single proximal cluster. Moving forward, multiple proximal clusters need to be connected to each other via their mediator nodes as explained in Section III. Multiple options can be explored in implementing this.
   (a) Each mediator node can run another instance of ZK by constructing a ZK ensemble at the mediator cluster layer. The advantage of this approach is that whole of Astro will be based on ZK, making it easy to maintain. The main drawback here is that all nodes in ZK communicate with each other in a peer-to-peer fashion, which leads to unncessary communication overhead. Also, the two instances of ZK running on the mediator (one belonging to the proximal cluster and the other belonging to the mediator cluster) would lead to processing overhead on the mediator device – a new communication protocol needs to be designed between these two instances of ZK within the mediator node.
   (b) Distributed Hash Tables (DHTs) can be leveraged to reduce the communication overhead and efficient lookup across proximal clusters [14], [15]. At the mediator cluster layer, there is no need for the data to be replicated and evenly distributed across all mediator nodes and DHTs serve this purpose by storing only necessary information needed for efficient hopping or lookup.
3) *Other minor enhancements*: Follower GC is important for supporting mobility of devices across proximal clusters. Support for follower GC needs to be enabled. Astro has been tested assuming steady users periodically requesting for resources and deallocating resources after use. User crashes can be supported to make the system more robust. ZK clients are used across the system in multiple functionalities to interact with the cluster. This can be optimized further by having a global ZK client always running to avoid client start and stop latencies. This also helps preventing orphan client threads which remain operational due to crashes.
4) *Evaluations*: Current Astro experiments have been performed on a single system using multi-threading. However, the experiments also need to be run on multiple systems and on ensembles considering of multiple devices. Edge device infrastructures such as EdgeNet [31] can be effectively used for evaluating the system. More evaluations can be carried out by comparing the results with existing benchmarks.

IX. CONCLUSION

In this work, we have presented Astro, an efficient system that provides resource discovery, registry, sharing and access management in edge networks. First, The architecture is presented with local proximal clusters and an uber mediator cluster. Then, the current implementation design for the proximal cluster is elaborated – supports concurrent device joins and leaves, user requests, and device crashes. The stablity of the system is also ensured by relying on the WiFi AP to host the quorum and the leader for a reliable ZK ensemble. Finally, We evaluated the latencies of various functionalities, and presented our analyses of device and user lifetimes. Our observation of the two-phase commit algorithm also led to the work on an optimized approach that drastically reduces the communication overhead present in the current approach.

ACKNOWLEDGMENTS

This project is a part of CS7210 (Distributed Computing) Course at Georgia Institute of Technology offered by Prof. Ada Gavrilovska. We would like to thank Prof. Gavrilovska and Dr. Ketan Bharadwaj for their guidance throughout the project.

REFERENCES

[1] Rad, B. B., & Shareef, A. A. (2017). "Fog Computing: A Short Review of Concept and Applications." International Journal of Computer Science and Network Security, 17(11), 68-74.
[2] Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012, August). "Fog computing and Its Role In The Internet of Things." In Proceedings of the first edition of the MCC workshop on Mobile cloud computing (pp. 13-16). ACM.
[3] Yi, S., Li, C., & Li, Q. (2015, June). "A survey of fog computing: concepts, applications and issues." In Proceedings of the 2015 workshop on mobile big data (pp. 37-42). ACM.
[4] Apache ZooKeeper releases, https://zookeeper.apache.org/releases.html
[5] Hunt, Patrick, et al. "ZooKeeper: Wait-free Coordination for Internet-scale Systems." USENIX annual technical conference. Vol. 8. No. 9. 2010.
[6] Rafetseder, A., Phringer, L., & Cappos, J. (2017, October). "Practical fog computing with seattle." In Fog World Congress (FWC), 2017 IEEE (pp. 1-7). IEEE.
[7] Datta, S. K., Da Costa, R. P. F., & Bonnet, C. (2015, December). "Resource discovery in Internet of Things: Current trends and future standardization aspects." In Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on (pp. 542-547). IEEE.
[8] Rodrigues, R., Liskov, B., & Shrira, L. (2002, July). "The design of a robust peer-to-peer system." In Proceedings of the 10th workshop on ACM SIGOPS European workshop (pp. 117-124). ACM.

[9] Gupta, I., Birman, K., Linga, P., Demers, A., & Van Renesse, R. (2003, February). "Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead." In International Workshop on Peer-to-Peer Systems (pp. 160-169). Springer, Berlin, Heidelberg.
[10] Kazaa Media Desktop. http://www.kazaa.com.
[11] Gnutella. http://www.gnutella.com.
[12] Kaashoek, M. F., & Karger, D. R. (2003, February). "Koorde: A simple degree-optimal distributed hash table. In International Workshop on Peer-to-Peer Systems" (pp. 98-107). Springer, Berlin, Heidelberg.
[13] Mizrak, A. T., Cheng, Y., Kumar, V., & Savage, S. (2003, June). "Structured superpeers: Leveraging heterogeneity to provide constant-time lookup." In Proceedings the Third IEEE Workshop on Internet Applications. WIAPP 2003 (pp. 104-111). IEEE.
[14] Salem, A., Salonidis, T., Desai, N., & Nadeem, T. (2017, October). "Kinaara: Distributed discovery and allocation of mobile edge resources." In Mobile Ad Hoc and Sensor Systems (MASS), 2017 IEEE 14th International Conference on (pp. 153-161). IEEE.
[15] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., & Balakrishnan, H. (2003). "Chord: a scalable peer-to-peer lookup protocol for internet applications." IEEE/ACM Transactions on Networking (TON), 11(1), 17-32.
[16] Sun, Y., & Zhang, N. (2017). "A resource-sharing model based on a repeated game in fog computing." Saudi journal of biological sciences, 24(3), 687-694.
[17] Zhang, H., Zhang, Y., Gu, Y., Niyato, D., & Han, Z. (2017). "A hierarchical game framework for resource management in fog computing." IEEE Communications Magazine, 55(8), 52-57.
[18] Yu, R., Huang, X., Kang, J., Ding, J., Maharjan, S., Gjessing, S., & Zhang, Y. (2015). "Cooperative resource management in cloud-enabled vehicular networks." IEEE Transactions on Industrial Electronics, 62(12), 7938-7951.
[19] Wang, N., Varghese, B., Matthaiou, M., & Nikolopoulos, D. S. (2017). "ENORM: A framework for edge node resource management."IEEE Transactions on Services Computing.
[20] Gu, L., Zeng, D., Guo, S., Barnawi, A., & Xiang, Y. (2017). "Cost efficient resource management in fog computing supported medical cyber-physical system." IEEE Transactions on Emerging Topics in Computing, 5(1), 108-119.
[21] C. DSouza, G. Ahn, and M. Taguinod, "Policy-Driven Security Management for Fog Computing: Preliminary Framework and a Case Study," 15th IEEE Intl. Conf. Info. Reuse and Integration, IRI 2014, 2014, pp. 1623.
[22] S. Salonikias, I. Mavridis, and D. Gritzalis, "Access Control Issues in Utilizing Fog Computing for Transport Infrastructure," 10th Intl. Conf. Critical Info. Infrastructures Security, CRITIS 2015, ser. LNCS, vol. 9578, Springer, 2015, pp. 1526.
[23] I. Stojmenovic et al., "An Overview of Fog Computing and Its Security Issues," Concurrency and Computation: Practice and Experience, vol. 28, 2016, pp. 22913005.
[24] F. Li et al., "Robust Access Control Framework for Mobile Cloud Computing Network," Comp. Commun., vol. 68, 2015, pp. 6172.
[25] B. Zaghdoudi, H. K. Ayed, and W. Harizi, "Generic Access Control System for Ad Hoc MCC and Fog Computing," 15th Intl. Conf. Cryptology and Network Security, CANS 2016, ser. LNCS, vol. 10052. Springer, 2016, pp. 40015.
[26] M. B. Mollah, M. A. K. Azad, and A. Vasilakos, "Secure Data Sharing and Searching at the Edge of Cloud-Assisted Internet of Things," IEEE Cloud Computing, vol. 4, no. 1, 2017, pp. 3442.
[27] L. Popa et al., "Cloudpolice: Taking Access Control Out of the Network," 9th ACM Wksp. Hot Topics in Networks, 2010, pp. 16.
[28] ZooKeeper Dynamic Reconfiguration, https://zookeeper.apache.org/doc/r3.5.3-beta/zookeeperReconfig.html
[29] Shraer, A., Reed, B., Malkhi, D., & Junqueira, F. P. (2012, June). "Dynamic Reconfiguration of Primary/Backup Clusters." In USENIX Annual Technical Conference (pp. 425-437).
[30] M. Lenczner and A. G. Hoen, "CRAWDAD dataset ilesansfil/wifidog (v.2015-11-06)," Downloaded from http://tinyurl.com/zxorzl7, Nov. 2015.
[31] EdgeNet framework, http://edge-net.org/