Design of Distributed Calculation Scheme using Network Address Translation for Ad-hoc Wireless Positioning Network

Jumpei Kajimura*, Shigemi Ishida*, Shigeaki Tagashira**, and Akira Fukuda*

*ISEE, Kyushu University,
Motooka 744, Nishi-ku, Fukuoka-shi, Fukuoka, 819-0395 Japan
**Faculty of Informatics, Kansai University,
Ryozenji-cho 2-1-1, Takatsuki-shi, Osaka, 569-1095 Japan

Abstract. We have developed an ad-hoc wireless positioning network (AWPN) to realize on-demand indoor location-based services [10]. This paper extends our AWPN to handle huge number of localization requests. In AWPN, WiFi APs measure received signal strength (RSS) of WiFi signals and send the RSS information to a localization server via a WiFi mesh network. The maximum number of WiFi devices is therefore limited by computational resources on the localization server. We push this limit by introducing a new distributed calculation scheme: we use the MapReduce computation framework and perform map processes on APs and reduce processes on localization servers. We also utilize a network router capable of network address translation (NAT) for shuffle processes to provide scalability. We implemented and evaluated our distributed calculation scheme to demonstrate that our scheme almost evenly distributes localization calculations to multiple localization servers with approximately 26 % variations.

Keywords: Ad-hoc wireless positioning network (AWPN), MapReduce, distributed calculation, network address translation (NAT).

1 Introduction

In recent years, smartphones have become prevalent, which pushes increasing attention to location-based services. Location-based services are mainly developed for outdoor use because the global positioning system (GPS) is widely available in outdoor environments. Indoor localization is now more required to extend location-based services to indoor environments.

We are developing a WiFi ad-hoc wireless positioning network (AWPN) to realize on-demand indoor location-based services that are used in one-time use scenarios such as a navigation in an exhibition event. The AWPN is a localization system built on a WiFi mesh network. In AWPN, WiFi access points (APs) capture IEEE 802.11 `ProbeRequest` frames sent from a WiFi device and measure received signal strength (RSS) of the frames. The RSS-data is then sent to a localization server via a WiFi mesh network to estimate the device location.

When we use AWPN in a large indoor environment, the localization server receives huge number of RSS-data, which increases computational requirements on the localization server. Large-scale AWPN consists of many WiFi APs that receive signals from many WiFi devices. Although single localization calculation completes in few milliseconds [10], localization calculations for hundreds of WiFi devices require considerable time. Especially, smartphones send many `ProbeRequest` frames every second, which drastically increases the number of localization calculations.

To address the calculation load problem, the MapReduce distributed calculation systems [2, 3] such as Hadoop [21] have been widely adopted. MapReduce systems, however, are inefficient for AWPN localization calculations because computational resource for distribution is not negligible. MapReduce systems consists of three processes: *map* process in which calculation tasks are associated with specific hash values named keys, *shuffle* process in which map tasks are distributed to calculation nodes based on the keys, and *reduce* process in which mapped data are aggregated to calculate final results. The MapReduce effectively distributes calculation load onto calculation nodes when the map and reduce processes are heavier than the shuffle process. In AWPN, localization calculation itself is lightweight computation. Shuffle processes and data communications between calculation nodes for shuffling are not negligible in AWPN.

As a new solution for the calculation load problem in AWPN, this paper presents a distributed calculation scheme such that MapReduce processes are distributed to APs and localization servers. In the proposed calculation scheme, APs perform map processes and localization servers perform reduce processes. The APs measure the RSS of a `ProbeRequest` frame and determine the localization server to send the RSS-data based on the information in the `ProbeRequest` frame. RSS-data, generated on multiple APs, of an identical `ProbeRequest` frame is therefore collected on the same localization server.

Practically, we utilize a network router for shuffle processes to easily support scalability. The required number of localization servers is dependent on the scale of AWPN and the number of WiFi devices to be localized. To avoid reconfiguration of WiFi APs in the environment when the number of localization servers changes, we use network address translation (NAT) on a network router to forward RSS-data to localization servers. The router is specified as a default gateway in AWPN to collect all the RSS-data on the router. RSS-data is forwarded to a specific localization server based on a key value in the RSS-data. The number change of localization servers only requires reconfiguration of address translation rules, which is defined in the network router.

Note that our approach is another form of MapReduce implementation with a simple feature set. Our distributed calculation scheme does not provide features such as dynamic scaling and fault tolerance that are widely available in original MapReduce systems. These features are often insignificant in localization systems for location-based services.

To demonstrate the effectiveness of the proposed distributed calculation scheme, we conducted experimental evaluations in a Kyushu University building.

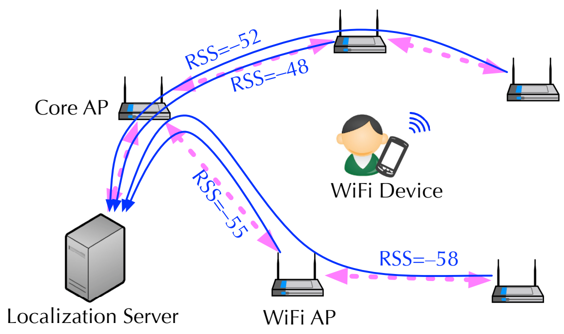


Fig. 1. Overview of ad-hoc wireless positioning network (AWPN)

The experimental evaluations reveal that the proposed distributed calculation scheme successfully distributed RSS-data to localization servers with imbalance of 26 %.

The remainder of this paper is structured as follows. Section 2 briefly describes AWPN and shows requirements of a distributed calculation scheme in AWPN. In Section 3, we present a distributed calculation scheme using network address translation for AWPN, followed by implementation in Section 4. In Section 5, we conducted experimental evaluations of the proposed distributed calculation scheme. Finally, Section 6 concludes the paper.

2 Distributed Calculation in Ad-hoc Wireless Positioning Network

2.1 Ad-hoc Wireless Positioning Network

Ad-hoc Wireless Positioning Network (AWPN) is a WiFi mesh network capable of localizing WiFi devices [17]. Figure 1 depicts an overview of AWPN. To construct AWPN, we install multiple WiFi APs into a localization target area and connect a localization server to an AP named a core AP. The network is automatically constructed with multi-hop links between APs. APs detect a WiFi signal sent from a WiFi device in the localization target area and measures received signal strength (RSS) of the signal. The RSS-data and the WiFi device address are then sent to a localization server. The localization server calculates the device location using multilateration with the RSS-data sent from multiple APs.

In AWPN, calculation load becomes heavier as the number of RSS-data increases because the localization server performs all the calculations. Distributed calculation is an effective solution to address this calculation load problem.

There are two requirements for distributed calculation in AWPN.

The first requirement is independence between AWPN scale and system configurations. When we extend a localization target area, we need to add APs and

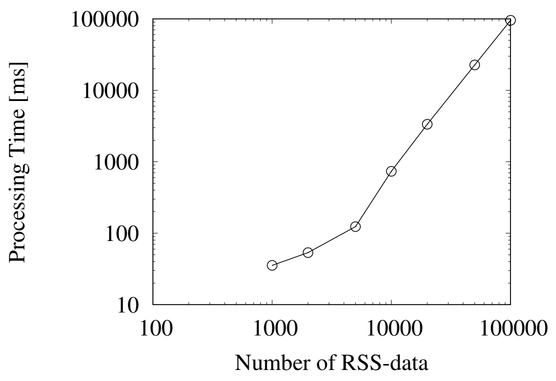


Fig. 2. RSS-data processing time on MongoDB

localization servers to process increased number of RSS-data. Changing system configurations such as AP configuration parameters, AP firmware, and localization server program requires much cost because there are hundreds of APs and localizations servers in AWPN.

The second requirement is small overhead. In large-scale AWPN, localization servers perform huge number of localization calculations because many WiFi devices transmit `ProbeRequest` frames more than once per second. Localization calculation is not a heavy task and is finished in few milliseconds [10]. Overhead for calculations including RSS-data reception should be minimized.

Figure 2 shows the time required for RSS-data processing on MongoDB distributed database [18]. We collected RSS-data using AWPN installed in our laboratory and inserted the RSS-data into MongoDB. On MongoDB, we grouped RSS-data by sender WiFi devices and counted the number of RSS-data in each group. We repeated data processing for 100 times and averaged the processing time. Figure 2 indicates that the processing time greatly increased as the number of RSS-data increased when the number of RSS-data was more than 5,000. We only counted the number of RSS-data in this example. We can confirm that overhead to retrieve data from database is considerable when we process huge number of RSS-data.

2.2 Related Works

For many high computation applications, MapReduce-based distributed calculation systems [2, 3] are widely adopted to process high volume of data. For example, the MapReduce systems are utilized in machine learning as well as data mining [9, 8, 15], clustering [22], pairwise document similarity calculation [6], and genome analysis [16]. Also there are many MapReduce extensions such as MRPGA [13], Twister [4], DELMA [7], Tiled-MapReduce [1], SpatialHadoop [5], and epiC [12].

The MapReduce systems, however, suffer from high overhead for localization calculation in AWPN because localization calculation is a lightweight computational task. In the MapReduce systems, RSS-data is first stored in a distributed database. RSS-data is then analyzed and grouped by sender devices in map processes to calculate device location. In reduce processes, device location is calculated using the grouped RSS-data. A node called a master node distributes map and reduce processes to calculation nodes. In AWPN, the load of the master node becomes significant when processing huge number of RSS-data. Data reading from the distributed database in map and reduce processes is another overhead in AWPN because results of map and reduce processes are stored in distributed nodes.

Kafka [14] is a distributed messaging system for realtime data processing, which is another form of distributed calculation systems. In Kafka, producers generate messages and send the messages to servers named brokers, which provide distributed data queues. Application servers, named consumers, retrieve messages from brokers at their own rate to process the messages. When we apply Kafka to AWPN localization calculations, APs send RSS-data to brokers and localization servers consume the RSS-data. Using a gateway service, Kafka easily adapts to the change of the number of localization servers. This publish/subscribe model is also used in several IoT middlewares such as DDS [19] and DPWS [11].

Although distributed data processing schemes using a publish/subscribe model can perform localization calculations with high flexibility, a broker requires higher computational resources compared to our approach. Data storage on a broker is also required to safely process stream data. Our approach only requires network routers with sufficient network capacity.

3 Distributed Calculation Scheme for AWPN

3.1 Overview

Figure 3 shows an overview of our distributed calculation scheme using address translation. Our key idea is to distribute MapReduce processes to APs, network router, and localization servers. An AP receives a `ProbeRequest` frame and measures received signal strength (RSS) of the frame. The AP performs a map process; the AP calculates a key value based on the information in the `ProbeRequest` frame. The RSS value as well as key value is sent to a network router as RSS-data. When the network router receives RSS-data, the router performs a shuffle process; RSS-data is sent to the localization server associated with the key value in the RSS-data. The RSS-data with the same key values is therefore collected to the same localization server. The localization server then performs a reduce process, i.e., calculates location of a WiFi device.

Following subsections describe details of map and shuffle processes.

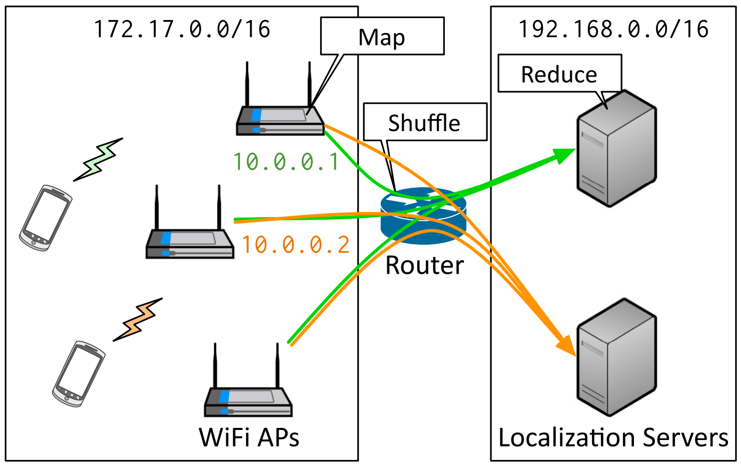


Fig. 3. Overview of distributed calculation scheme using network address translation for AWPN

3.2 Map Process

In a map process, an AP calculates a key value based on the information in a `ProbeRequest` frame sent from a WiFi device. An AP retrieves the information below to calculate a key value:

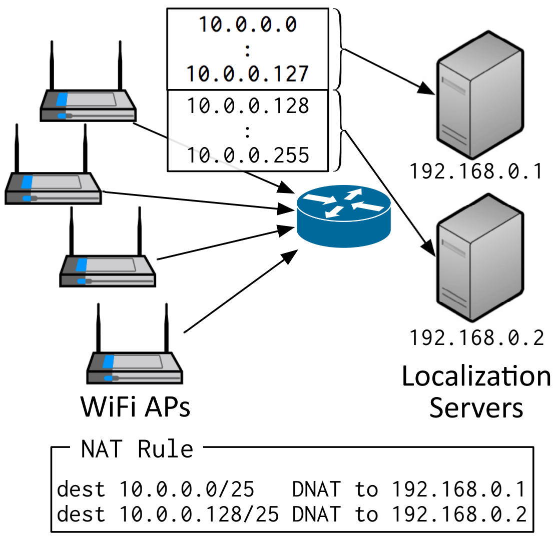
– The MAC address of a source WiFi device
– The `sequence` number of a `ProbeRequest` frame
– The reception time of a `ProbeRequest` frame

The key values are associated with IP addresses in an address space not used in a WiFi mesh network. An AP sends RSS-data to the IP address associated with a calculated key value. We configure the WiFi mesh network to use a network router as a default gateway. All the RSS-data is therefore sent to the network router. In Fig. 3, for example, RSS-data sent to an address in `10.0.0.0/24` address space is actually sent to the network router because `10.0.0.0/24` is outside of `172.17.0.0/16` network.

Key value calculation algorithm should be simple enough because APs have limited computational resources. As a simple example in this paper, a key value k is calculated from the last byte m of the MAC address of a source WiFi device and the `sequence` number s of a `ProbeRequest` frame as

$$k = (m + s) \mod 256. \tag{1}$$

`Sequence` number should be included in a key calculation because a WiFi device successively sends `ProbeRequest` frames with different `sequence` numbers in a short time. For binding between key values and IP addresses, we map an 8-bit key value to a last byte of an IP address in `10.0.0.0/24` address space.



```
dest 10.0.0.0/25   DNAT to 192.168.0.1
dest 10.0.0.128/25 DNAT to 192.168.0.2
```

Fig. 4. Overview of shuffle process in network router

3.3 Shuffle Process

In a shuffle process, a network router changes the destination address of RSS-data using network address translation (NAT) to forward RSS-data to localization servers. The network router is responsible for distribution of RSS-data for localization calculation. The number change of localization servers only requires reconfiguration of the network router.

Figure 4 shows an overview of a shuffle process in a network router. The figure shows an example with two localization servers `192.168.0.1` and `192.168.0.2`. APs send RSS-data to an IP address in `10.0.0.0/24` address space. The destination address space `10.0.0.0/24` is divided into `10.0.0.0/25` and `10.0.0.128/25` subnets, each of which is assigned to a localization server. The network router performs network address translation with rules shown in Fig. 4 to forward RSS-data to localization servers. Although we can divide the destination address space at any point, the size of subnets should be the same to evenly distribute RSS-data to localization servers.

When the number N of localization servers is not the power of 2, i.e., $N \neq 2^n$ (n is zero or a positive integer), we need complicated address translation rules. For example, when we add another localization server in Fig. 4, we want to evenly divide the address space into three subnets below:

– `10.0.0.0` $\sim$ `10.0.0.84`,
– `10.0.0.85` $\sim$ `10.0.0.169`,
– `10.0.0.170` $\sim$ `10.0.0.255`.

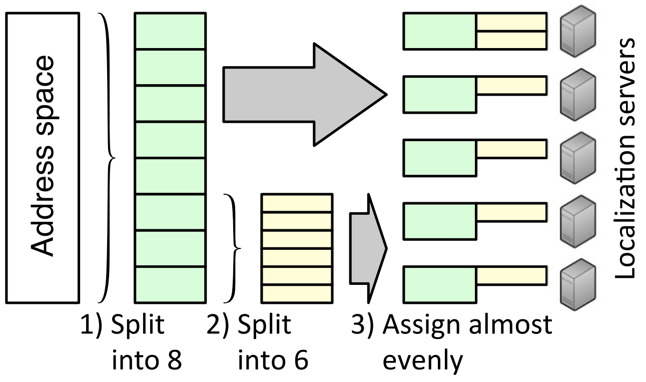


Fig. 5. Example of address space division ($N = 5$, $k_{max} = 2$)

The size of a subnet is 85 or 86 in this case. Subnets are defined by a netmask, which restricts the size of a subnet to the power of 2. We cannot evenly divide the address space into three subnets with any netmask.

When $N \neq 2^n$, we divide an address space into subnets until the number of the subnets is greater than N and assign the subnets to localization servers. Here we explain the address division using an example when the number N of localization servers is five as shown in Fig. 5. We first divide the address space into eight subnets. Five of the subnets are assigned to each localization server. Three remaining subnets are more divided into six small subnets, five of which are assigned to localization servers. We repeat this division process for up to k_{max} times and assign remaining subnets to localization servers as shown in Fig. 5.

Although the address division process results in imbalance of the number of addresses in subnets, the imbalance decreases as the maximum number k_{max} of divisions increases. Figure 6 shows address imbalance B_{addr} as a function of the number N of localization servers. Address imbalance B_{addr} is defined as

$$B_{addr} = \max \left| \frac{a_i - \bar{a}}{\bar{a}} \right|, \tag{2}$$

where a_i is the number of addresses assigned to localization server i and $\bar{a}$ is the average number of addresses assigned to localization servers. Address imbalance B_{addr} becomes 0 for an ideal case, i.e., addresses are evenly assigned to localization servers. Figure 6 indicates that the maximum address imbalance decreases as 0.875, 0.25, 0.125, and 0.063 as the maximum number k_{max} of divisions increases from 1 to 4. Increase in k_{max} results in increase in the computation load on a network router. k_{max} is determined based on computational resources on the router.

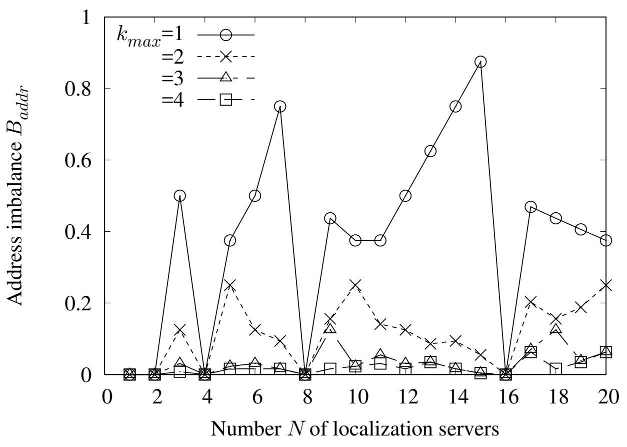


Fig. 6. Address imbalance B_{addr} as a function of the number N of localization servers ($k_{max} = 1, 2, 3, 4$)

Table 1. Specifications of PCWL-0100 [20]

Range in line-of-sight	Approx. 150 m
TX power of mesh wireless	16 dBm
TX power of access wireless	16 dBm
Access wireless standard	IEEE 802.11b/g
Number of mesh wireless IFs	2 (except an access wireless IF) 5.15 ∼ 5.35 GHz
Physical dimensions	W142 mm×H118 mm×D39 mm
Weight	450 g

4 Implementation

We implemented the proposed distributed calculation scheme using off-the-shelf devices. Figure 7 shows the overview of our implementation. We installed four PicoCELA PCWL-0100 APs and a Netgear WNDR4300 network router, which are shown in Fig. 8. Table 1 shows specifications of PCWL. PCWLs are WiFi APs that automatically build a WiFi mesh network. We implemented a C program that captures `ProbeRequest` frames to generate RSS-data and perform map processes on Linux running on PCWL.

We prepared three virtual machines: two for localization servers and one for a database server. The virtual machines were managed by the VMware ESXi 6.0 hypervisor running on a Supermicro 6018R-TD server with a 1.8 GHz eight-core Intel XeonE5-2630Lv3 CPU, 16 GB memory, and four 2 TB disk drives. Each virtual machine used single CPU core and 2 GB memory. Debian/GNU Linux 8.0 was running on virtual machines, which were built on separate disk drives to minimize mutual influence between the virtual machines.

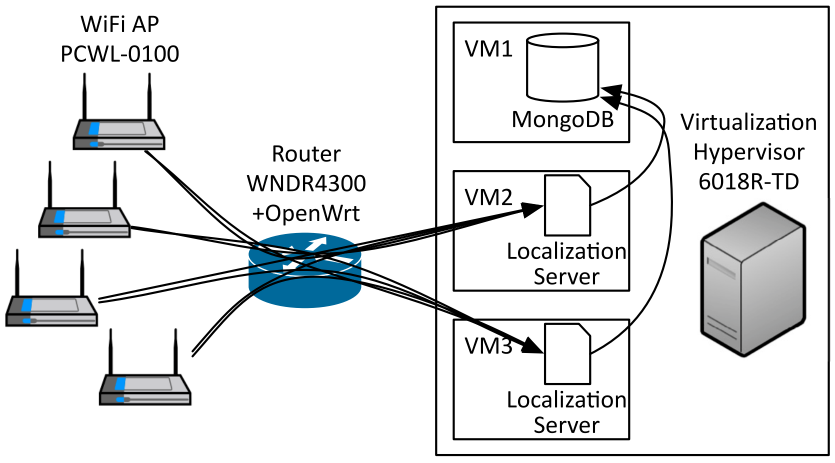


Fig. 7. Overview of implementation

The localization server was implemented as a C++ program. The localization server received RSS-data from the network router and estimate WiFi device location using a simple multilateration algorithm. The results were sent to a MongoDB database server.

5 Evaluation

To validate the effectiveness of the proposed distributed calculation scheme presented in Section 3, we evaluated RSS-data imbalance that indicates fairness of RSS-data distribution. We also evaluated the CPU usage and time for localization calculations as a function of RSS-data traffic to estimate the number of localization servers required for practical deployment.

In our evaluations, we used RSS-data collected in a real environment. We installed four PCWLs in our laboratory and collected RSS-data generated from `ProbeRequest` frames sent from user devices such as smartphones and laptops. The RSS-data was collected for approximately three and half days. The number of collected RSS-data is 137,061.

5.1 RSS-Data Imbalance

RSS-data imbalance is a figure that indicates how uniformly RSS-data is distributed to localization servers. RSS-data imbalance B_{rss} is defined in the same manner as the address imbalance as

$$B_{rss} = \max \left| \frac{r_i - \bar{r}}{\bar{r}} \right| , \tag{3}$$

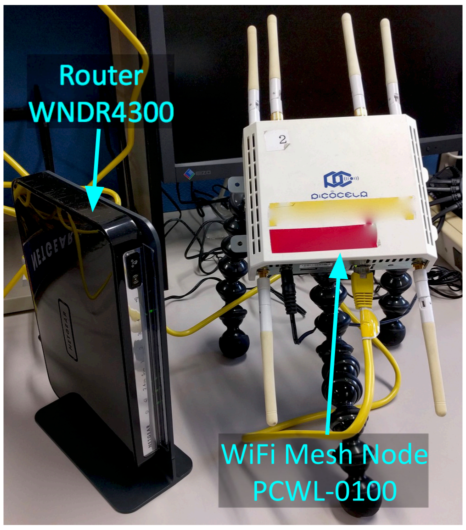


Fig. 8. AP and network router used in implementation

where r_i is the number of RSS-data received on localization server i and $\bar{r}$ is the average number of RSS-data received on localization servers. RSS-data imbalance B_{rss} becomes 0 for an ideal case, i.e., RSS-data is evenly distributed to localization servers.

We calculated the number of RSS-data received on each localization server using the RSS-data collected in a real environment, while changing the number of localization servers. For each RSS-data, destination localization server was calculated using the map process described in Eq. (1) and the shuffle process presented in Section 3.3. RSS-data imbalance was calculated using Eq. (3) with the number of RSS-data received on each localization server.

Figure 9 shows RSS-data imbalance B_{rss} as a function of the number N of localization servers. Figure 9 indicates the following:

– Comparing Figs. 6 and 9, RSS-data imbalance B_{rss} curve is similar to the address imbalance curve in Fig. 6. When address imbalance was big, RSS-data was not evenly distributed to localization servers, resulting in big RSS-data imbalance.
– RSS-data imbalance B_{rss} had a tendency to decrease as the maximum number k_{max} of divisions increases. In a range of k_{max} from 1 to 4, RSS-data imbalance B_{rss} became maximum at 0.26 when $k_{max} = 3$.

From the above results, we conclude that a hash function used in a map process is a key factor to evenly distribute RSS-data in our distributed calculation scheme. The simple hash function presented in Eq. (1) in Section 3.2 exhibited low performance in terms of fair distribution of RSS-data.

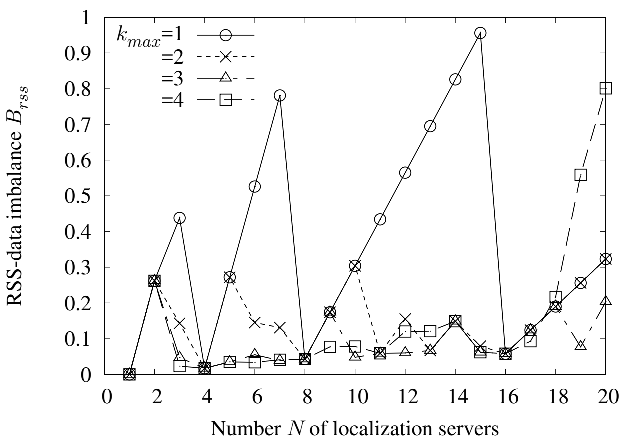


Fig. 9. RSS-data imbalance B_{rss} as a function of the number N of localization servers ($k_{max} = 1, 2, 3, 4$)

5.2 CPU Usage

To validate that a localization calculation is not a heavy task compared to RSS-data reception, we evaluated CPU usage while changing RSS-data traffic, i.e., the number of RSS-data sent in one second. We sent dummy RSS-data at a specific rate and recorded average CPU usage for 300 seconds using `sysstat` command. Not to perform wasting calculations, we only sent RSS-data that can be successfully localized. We compared CPU usages with and without localization calculations.

Figure 10 shows CPU usage as a function of RSS-data traffic. Figure 10 indicates the following:

– CPU usage almost linearly increased as the RSS-data traffic increased. The numbers of RSS-data receptions and localization calculations are proportional to RSS-data traffic, which linearly increased CPU usage.
– There was a slight difference of CPU usages between with and without localization calculations. The difference of CPU usages between with and without localization calculations, i.e., CPU usage for localization calculations, was quite small.

The above results reveal that localization calculation is a lightweight task in comparison with RSS-data reception.

5.3 Calculation Time

To estimate how many servers are required to localize huge number of WiFi devices, we evaluated localization calculation time. We modified localization calculation program on a localization server installed in a real environment to record

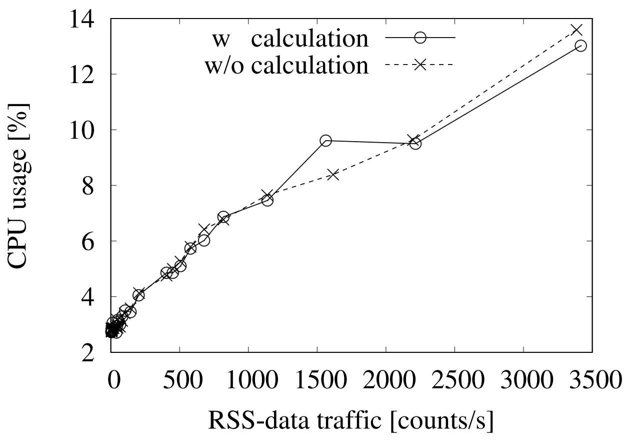


Fig. 10. CPU usage as a function of RSS-data traffic

start and end time for every localization calculation. Localization calculations were performed for 80,941 times with 137,061 RSS-data.

Figure 11 shows an empirical cumulative distributed function (ECDF) of localization calculation time. Black and blue lines in the Fig. 11 show the results for all calculations and successful calculations, respectively. Figure 11 shows the following:

- More than 80 % of calculations were completed in 5 milliseconds. Localization calculation is not a heavy task for a localization server and doesn't take much time for a single calculation.
- 83.6 % of successful calculations were completed in 5 milliseconds. 83.6 % and 96.9 % of successful calculations were completed in 5 and 10 milliseconds, respectively. For more than 100 localization calculations per second, multiple servers or multi-thread programming is required to complete localizations in realtime.

6,955 localization calculations have been succeeded, which is 8.6 % of all the calculations. We used four APs in this evaluation and calculated device location using multilateration. Multilateration requires RSS-data from all the four APs to estimate device location in our evaluation environment. 69.3 % of localization calculations failed because of less number of RSS-data, which completed in 5 milliseconds. Remainder 22.1 % of calculations diverged because of the variations of RSS caused by multi-paths and measurement errors, which took longer time.

The above results reveal that our distributed calculation scheme requires parallel computation for more than 100 localization calculations per second. Referring to Section 5.2, the number of localization servers might be estimated based on the CPU usage of RSS-data reception. As shown in Fig. 10, the CPU usage linearly increases as RSS-data traffic increases. We might need more than two servers when RSS-data traffic is more than 25,000 counts every second.

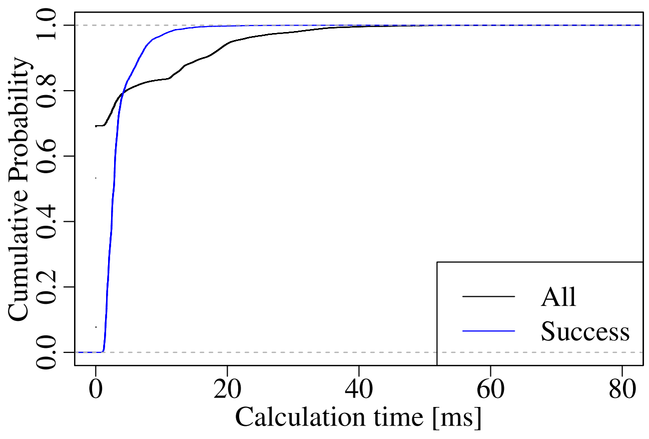


Fig. 11. Empirical cumulative distributed function (ECDF) of localization calculation time for all and successful calculations

The RSS-data traffic greatly increases as the numbers of WiFi devices and APs increase.

6 Conclusion

This paper presented a new distributed calculation scheme for an ad-hoc wireless positioning network (AWPN) to process huge number of localization requests. Our approach is to distribute MapReduce processes to WiFi APs, a network router, and localization servers: map processes on APs, shuffle processes on a router, and reduce processes on localization servers. Using network address translation (NAT) in shuffle processes, our distributed calculation scheme easily provides scalability with variable number of localization servers. We conducted experimental evaluations in a real environment and confirmed that our scheme successfully distributed localization calculations with the maximum RSS-data imbalance of 0.26. The RSS-data imbalance of 0.26 might be insufficient in practical use cases. We are working on a hash function in a shuffling process to evenly distribute calculation tasks.

Acknowledgments. This work was supported in part by JSPS KAKENHI Grant Numbers 15H05708, 15K12021, 16K16048, and 17H01741, and the Cooperative Research Project of the Research Institute of Electrical Communication, Tohoku University.

References

1. Chen, R., Chen, H.: Tiled-MapReduce: Efficient and flexible MapReduce processing on multicore with tiling. ACM Trans. on Architecture and Code Optimization (TACO) 10(1), 3:1–3:30 (Apr 2013), article no.3

2. Dean, J., Ghemawat, S.: MapReduce: Simplified data programming on large clusters. Commun. ACM 51(1), 107–113 (Jan 2008)
3. Dean, J., Ghemawat, S.: MapReduce: A flexible data processing tool. Commun. ACM 53(1), 72–77 (Jan 2010)
4. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.H.: Twister: A runtime for iterative MapReduce. In: Proc. ACM Int. Symp. on High Performance Distributed Computing (HPDC). pp. 810–818 (Jun 2010)
5. Eldawy, A.: SpatialHadoop: Towards flexible and scalable spatial processing using MapReduce. In: Proc. ACM SIGMOD PhD Symp. pp. 46–50 (Jun 2014)
6. Elsayed, T., Lin, J., Oard, D.W.: Pairwise document similarity in large collections with MapReduce. In: Proc. ACL, Human Language Technologies: Short Papers (HLT-Short). pp. 265–268 (Jun 2008)
7. Fadika, Z., Govindaraju, M.: DELMA: Dynamically ELastic MApReduce framework for CPU-intensive applications. In: Proc. IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid). pp. 454–463 (May 2011)
8. Ghoting, A., Kambadur, P., Pednault, E., Kannan, R.: NIMBLE: A toolkit for the implementation of parallel data mining and machine learning algorithms on MapReduce. In: Proc. ACM KDD. pp. 334–342 (Aug 2011)
9. Ghoting, A., Krishnamurthy, R., Pednault, E., Reinwald, B., Sindhwani, V., Tatikonda, S., Tian, Y., Vaithyanathan, S.: SystemML: Declarative machine learning on MapReduce. In: Proc. IEEE Int. Conf. on Data Engineering (ICDE). pp. 231–242 (Apr 2011)
10. Ishida, S., Tagashira, S., Arakawa, Y., Fukuda, A.: On-demand indoor location-based service using ad-hoc wireless positioning network. In: Proc. IEEE Int. Conf. on Embedded Software and Systems (ICESS). pp. 1005–1013 (Aug 2015)
11. Jammes, F., Mensch, A., Smit, H.: Service-oriented device communications using the devices profile for web services. In: Proc. ACM Int. Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC) (Nov–Dec 2005)
12. Jiang, D., Wu, S., Chen, G., Ooi, B.C., Tan, K.L., Ku, J.: epiC: an extensible and scalable system for processing big data. The VLDB J. 25(1), 3–26 (Feb 2016)
13. Jin, C., Vecchiola, C., Buyya, R.: MRPGA: An extention of MapReduce for parallelizing genetic algorithms. In: Proc. IEEE Int. Conf. on eScience. pp. 214–221 (Dec 2008)
14. Kreps, J., Narkhede, N., Rao, J.: Kafka: a distributed messaging system for log processing. In: Proc. Int. Workshop on Networking Meets Databases (NetDB). pp. 1–7 (Jun 2011)
15. Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J.M.: Distributed GraphLab: a framework for machine learning and data mining in the cloud. In: Proc. Int. Conf. on Very Large Scale Data Bases (VLDB). pp. 716–727 (Aug 2012)
16. McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., DePristo, M.A.: The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. Genome Research 20(9), 1297–1303 (Sep 2010)
17. Miwa, N., Tagashira, S., Matsuda, H., Tsutsui, T., Arakawa, Y., Fukuda, A.: A multilateration-based localization scheme for adhoc wireless positioning networks used in information-oriented construction. In: Proc. IEEE Int. Conf. on Advanced Info. Networking Applications (AINA). pp. 690–695 (Mar 2013)
18. MongoDB, Inc.: MongoDB, `https://www.mongodb.com/`
19. Object Management Group: The OMG data-distribution service for real-time systems (DDS), `http://portals.omg.org/dds/`

20. PicoCELA: PCWL-0100 catalog. available online, `http://www.picocela.com/`
21. The Apache Software Foundation: Apache Hadoop, `http://hadoop.apache.org/`
22. Zhao, W., Ma, H., He, Q.: Parallel k-means clustering based on MapReduce. In: LNCS. vol. 5931, pp. 674–679 (Dec 2009), proc. Int. Conf. on Cloud Computing (CloudCom)