Lin Lin CS237 – Distributed Systems Middleware

Zerui Li

Michael Salviani

**Project Report**

**I. Objectives**

Our team will be investigating the use of an additional broker (VerneMQ) to the standard Big Active Data (BAD) publication/subscription model. The standard BAD model is diagramed below:

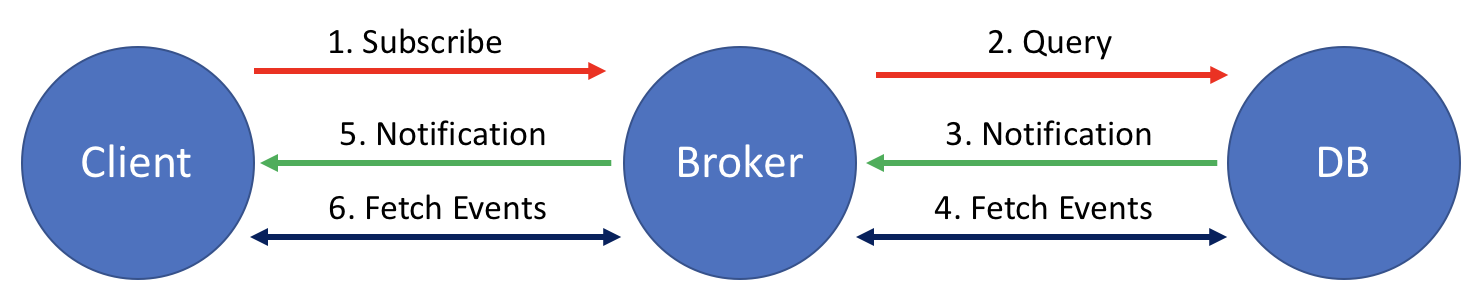


Figure 1: Standard Pub-Sub Model

This model includes clients, a broker they communicate with, and a backend database that receives and stores publisher material. The typical communication process for this model includes:

1. The client sends a subscription request to the broker
2. The broker sends a query to the DB to see if any events match the request of the client
3. The DB replies with a notification letting the broker know if there are events that match
4. The broker checks to make sure the client still has an active session, then fetches the event data from the DB
5. The broker sends a notification to the client that there is event data available
6. The client then fetches the events

This model works well but puts a lot of responsibilities on the broker. Our team investigated a new system that broke the responsibilities of the broker into two separate processes. The modified pub-sub model adds an additional broker (VerneMQ). This model changes the interaction with the client in two ways. The client needs to communicate with two brokers vs. one and the messages are pushed to the client instead of the notification/fetch system present in the standard model. See the next page for a diagram.

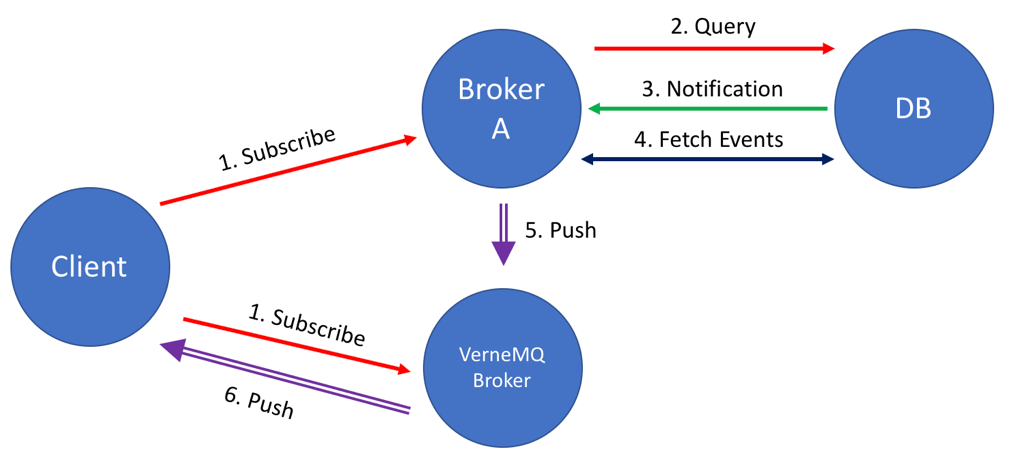


Figure 2: Modified Pub-Sub Model

The communication process for this model includes:

1. The client sends a subscription to both brokers
2. Broker A sends a query to the DB to see if any events match the request of the client
3. The DB replies with a notification letting the broker know if there are events that match
4. Broker A immediately fetches the event data from the DB
5. Broker A then pushes the event data to the VerneMQ Broker
6. The VerneMQ Broker pushes the event data to the client

Our objective is to find out if the addition of the VerneMQ broker adds any benefit (reliability, efficiency, etc.) to the system. We would assume that breaking the workload of a single broker into two would have pros and cons.

Pros

More throughput at each broker node

Nodes can be designed/chosen to be more efficient with specific responsibilities tasked with

With the addition of a push mechanism, messages should flow to the clients quicker

Cons

Since the new broker is being pushed messages, it would need to have local storage if a client is offline

Adding another node increases the surface area for bugs, communication errors, faults, and outages

The client now needs to communicate with two brokers instead of one

**II. Related Efforts**

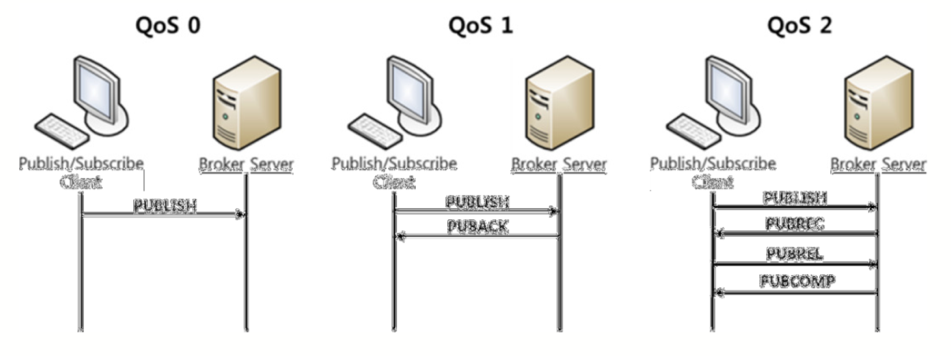
Most of the current research studies messaging protocols and their efficiency in various environments. For example, AMQP [4] has been proven [1, 5] to be very efficient and reliable in unstable, mobile networks where clients are connecting and disconnecting often. It is also well suited for use in the financial sector because it provides encryption [6]. MQTT uses Quality of Service levels [3] to vary the level of reliability of message delivery in a lightweight implementation for low bandwidth, high-latency environments [4]. One interesting facet of having a variety of protocols is that each has certain strengths. Those strengths can be used to choose the correct protocol for the environment that the system is operating in. For the most part, environments are fairly static. These systems, used inside of a company, on a specific network, will allow efficiencies. If the environment includes many wireless devices that join and leave the network frequently, a broker using a certain protocol can be used to best accommodate this. If the environment is the Internet with users around the world all getting data on a variety of devices, a more robust broker/protocol system can be put into place that is designed for that situation. These systems also need to consider the synchronization between the brokers and clients. If you want to offer, for instance, time decoupled services, the broker will need to be able to store data and have policies in place to deal with disconnected clients.

Figure 3: MQTT Quality of Service Levels [2]

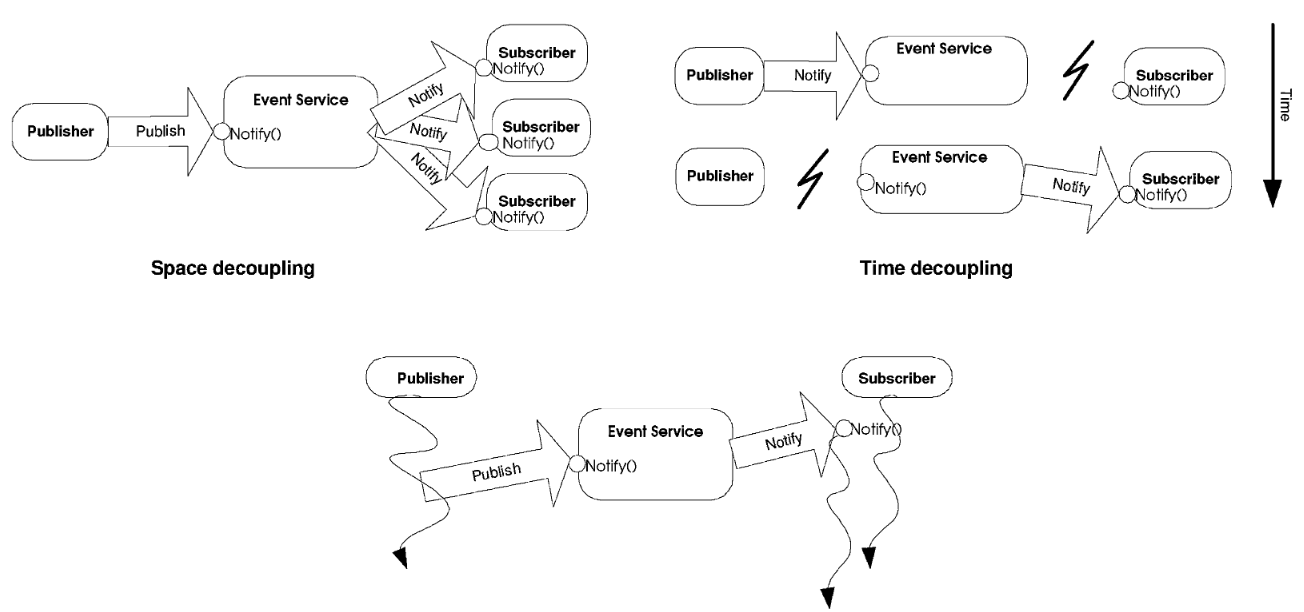


Figure 4: Synchronization Decoupling [7]

Scalability is also an issue. How well does the broker/protocol scale to sizes from a small company to clients in the millions? Scalability and reliability are often at odds with each other as well. P. Eugster et al [7] state:

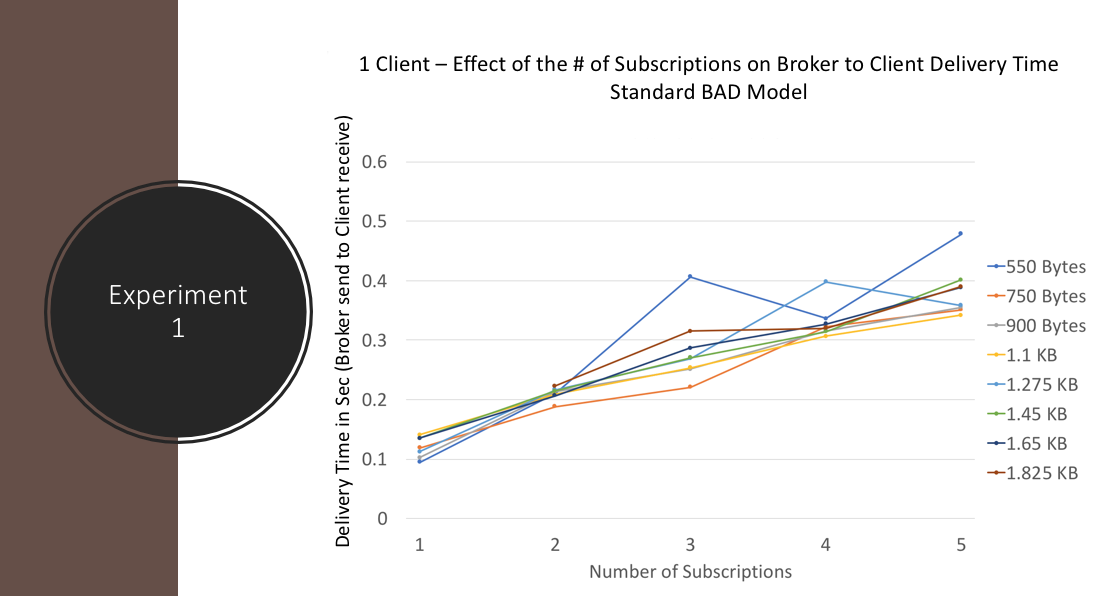
Scalability also often conflicts with other desirable properties. For instance, highly expressive and selective subscriptions require complex and expensive filtering and routing algorithms, and thus limit scalability. Similarly, strong reliability guarantees involve important overheads, because events must be logged, and missed events must be detected and retransmitted. Even protocols developed especially for wide-area networks, such as the sender-reliable Reliable Multicast Transport Protocol (RMTP), do not scale well to large numbers of subscribers because of the considerable amount of traffic resulting from message acknowledgments.

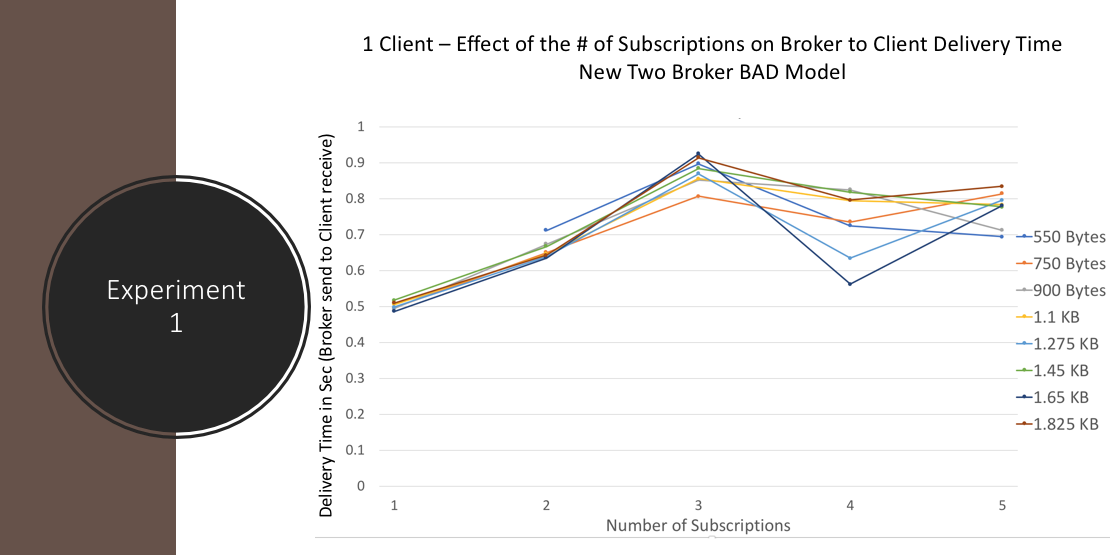
Some solutions to this scalability problem exist including Zhao et al’s paper that discusses the use of virtual time vectors to maintain temporal consistency in propagating subscriptions to clients [8].

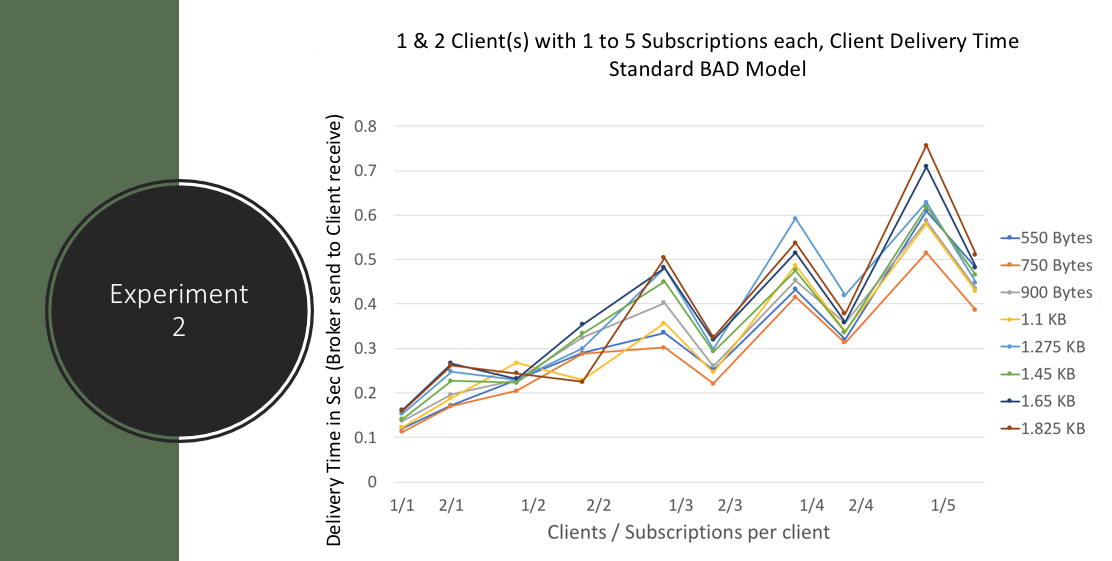
Our experiments look at the performance difference between two different broker structures to determine if there are any efficiencies that can be gained by specializing specific brokers to specific tasks.

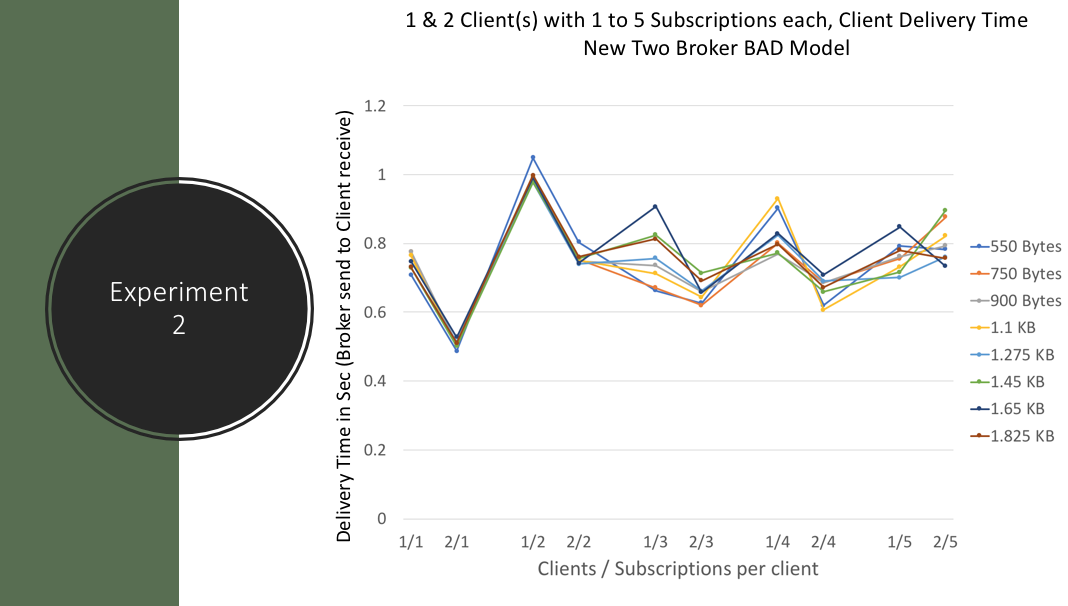
**III. Experiment**

As shown in Figures 1 and 2, our experiment tests two different BAD pub-sub models to determine if there are any efficiency gains in adding a specific use broker (VerneMQ) to handle sending message data to clients.









**IV. Conclusion**

Experiment 1 - With a single client and up to 5 subscriptions, the added VerneMQ broker had worse performance vs the Standard Model. This result shows that under light load, the addition of an additional hop/broker will slow delivery time.

Experiment 2 – While the Standard BAD Model had increasing latency under heavy load, the Two Broker BAD Model showed constant performance. The addition of the VerneMQ Broker proved more efficient as load increased.

Our experiment did not include the delay or latency inside the broker in the standard BAD model or the VerneMQ broker in the new model. The overhead costs of two brokers versus one should be evaluated. One of the benefits of a two broker model is the division of labor/processing under high loads. We saw some of those benefits in our experiment looking only at delivery time and we expect even greater benefits when including processing overhead.

Future research in this area could include a number of different experiments. Brokers are sometimes designed with a specific purpose or efficiency that the creators want to maximize. If a pub/sub system could use brokers that are designed specifically for certain tasks versus using general purpose brokers, there may be efficiencies to be gained.

One possible continuation of this research could be in the dynamic use of brokers/protocols based on changes in the environment. As the network or environment, the publish-subscribe system is overlaid on changes, the brokers/protocols could dynamically change to best optimize efficiency and redundancy while allowing for scalability. Some research in this area includes Jaeger et al’s paper that proposes this type of dynamic system [9] as well as two by Baldoni et al [10, 11].

**References**

[1] Jorge E. Luzuriaga , Miguel Perez , Pablo Boronat, Juan Carlos Cano, Carlos Calafate , Pietro Manzoni. *A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks*, IEEE 12th Consumer Communications and Networking Conference, 2015

[2] Shinho Lee, Hyeonwoo Kim, Dong-kweon Hong, Hongtaek Ju, *Correlation Analysis of MQTT Loss and Delay According to QoS Level*, International Conference on Information Networking (ICOIN), 28-30 Jan. 2013, pp. 714-717.

[3] D. Thangavel, X. Ma, A. Valera, H. Tan, and C. K. Tan. *Performance evaluation of MQTT and CoAP via a common middleware*, IEEE 9th Int. Conf. ISSNIP, 2014, pp. 1–6

[4] L. Magnoni. *Modern messaging for distributed systems*, J. Phys., Conf. Ser., vol. 608, no. 1, p. 012038, 2015.

[5] J. L. Fernandes, I. C. Lopes, J. J. P. C. Rodrigues, S. Ullah, *Performance Evaluation of RESTful Web Services and AMQP Protocol,* Proc. 5th ICUFN, 2013, pp. 810–815

[6] V. John, X. Liu, *A Survey of Distributed Message Broker Queues,* arXiv:1704.00411 [cs.DC] 2017.

[7] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. *The Many Faces of Publish/Subscribe*. ACM Computing Surveys, 35(2):114–131, June 2003

[8] Zhao, Y., Sturman, D., Bhola, S.: Subscription propagation in highly-available publish/subscribe middleware. In: Proceedings of the ACM/IFIP/USENIX 6th International Middleware Conference (Middleware 2004). (2004)

[9] Jaeger, M.A., Parzyjegla, H., Muhl, G., Herrmann, K.: Self-organizing broker topologies for publish/subscribe systems. In: SAC 2007, pp. 543–550. ACM Press, New York (2007)

[10] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. A self-organizing crash-resilient topology management system for content-based publish/subscribe. In A. Carzaniga and P. Fenkam, editors, 3rd

International Workshop on Distributed Event-Based Systems (DEBS’04), Edinburgh, Scotland, UK, May

2004. IEEE.

[11] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Subscription-driven self-organization in content-based publish/subscribe. Technical report, DIS, Mar. 2004