Edge-cloud hybrid model for distributed apps

Albert van der Linde
a.linde@campus.fct.unl.pt
NOVA LINCS & DI-FCT-UNL

ABSTRACT

The focus of this thesis is on bringing application logic closer to the client to enhance user experience. Previous work [1], has shown that by adding peer-to-peer connections between client devices, it is possible to obtain improvements in terms of user-to-user latency and server bandwidth, but the proposed approach is not adequate for every application, since some use cases require stronger consistency semantics (stronger than causal consistency) or may be required to deal with misbehaving users. The first challenge is on adding dynamic replication to both logic and data planes of the centralized component, to leverage on servers closer to end-users. The next challenge is creating decentralized support for coordination between devices (for stronger application semantics) and better security mechanisms to prevent and detect cheating (trust in decentralized systems).

1 INTRODUCTION

Many applications are made to serve content to users, or are focussed on enabling direct interactions between users. Despite these applications being user-centric, often being fully dedicated to enable interaction between users, the architectural model used to create these applications is based on a complete separation between client and server side. The client-to-server interaction model is traditionally used where client devices interact with a centralized component, which mediates all interactions between devices (i.e., users).

This architectural model brings disadvantages not only for the end user (of the application), but also to application providers (those who develop and/or monetize it). End users suffer from: (i) high interaction latency between clients, as the data-route is client-server-client; (ii) no interaction between clients when a server is unresponsive (or unreachable), which is notable when clients are close-by or sharing a local network. For applications developers and providers, issues are as follows: (i) a central point of failure/contention, as the central component is responsible to mediate all actions from every user, possibly to all other users; (ii) high server upkeep costs and risk for mistakes when provisioning for user load which might have catastrophic effects business wise.

Poor application performance, or user-perceived latency, leads to compromising the image of the company operating the application, even making users believe a website might have compromised security[2]. When thinking about multi-player games over the internet, latencies nearing 80ms are noticeable to end users and over 150ms negatively affects gameplay[3].

It is not trivial to address these issues, especially if we take into account that users nowadays expect almost no latency whatsoever while the actual amount of users of an application can change by an order of magnitude overnight.

EuroDW'18, April 2018, Porto, Portugal
.

A step towards user-centric services. We proposed to extend user-centric Internet services with peer-to-peer interactions. We designed a framework called Legion [1], which enables client web applications to replicate data from servers, operate over these replicas locally, and synchronize these replicas directly among client devices using the server for durability.

Figure 1 shows results of running a web application developed with a traditional client-server model (**G**oogle **Drive** **R**eal**T**ime API) versus our own (Legion) with 4, 8 and 16 clients editing data, showing major impact in client-client latency[1]. Notice, as clients are equally distributed across two locations, local client-client latencies can be very low if we leverage on peer-to-peer connections.

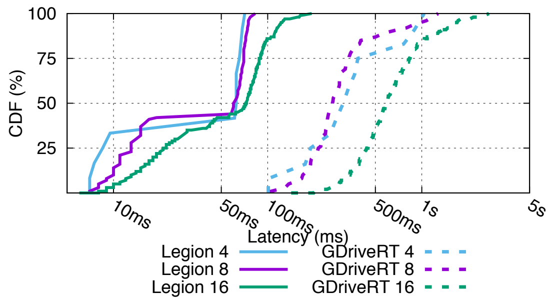

Figure 1: Update Propagation Latency

To further investigate the effects of our approach on end-user experience, we created a multi-player version of the popular Pacman game, where users control the Pacman and each ghost. The instance of the Pacman player serves as the owner of the game (i.e., it receives user input of all users, and applies them to the game). Other users receive updates to positions, directions, and map updates. Ghost users use interpolation to estimate where others are located, but in a highly interactive game this leads to displacement between the estimated and real position. Figure 2 shows this displacement using both systems. It is clear that adding peer-to-peer connections can greatly benefit the user experience in these applications.

This work also demonstrated a reduction in server network load, and support to continue operating over an already established network when the server becomes unavailable. Due to space restrictions we are unable to present these in detail. We refer the interested reader to [1].

Legion shows that some applications do indeed benefit from abandoning the traditional client-server model. A better application response time comes from locally executing operations. Reduced client-to-client latency comes from direct connections, sending operations to connected users directly instead of following the client-to-server-to-client route (which also enables user interaction

[1]The application used to compare both systems is exactly the same, changing only the script import headers of the HTML page.

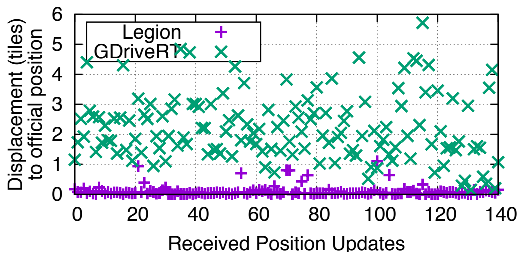

Figure 2: Multi-user Pacman

while disconnected from the server). A reduction in server load was possible to obtain as application instances that are connected to each other no longer need to all be individually connected to the server. Server load is not only alleviated by a direct reduction in client connections, but also by efficiently aggregating operations from multiple clients (not just data compression, but summarizing groups of operations).

We claim that this hybrid interaction model where clients can naturally interact with each other while leveraging the server for durability and assisting in some key aspects of the system operation, is the correct approach for devising new user-centric applications.

1.1 Future work

Providing static content to a large number of users can be addressed through the usage of Content Distribution Networks, but it is unclear how to allow users to share application state, and especially, be able to modify that state efficiently.

An application developer has to make sure that not only all clients are able to operate over the data (execute read and write operations) and that clients are updated when data of their interest changes, but also that the supporting system itself can cope with an increasing write load with an increasing number of clients while delivering, in an interactive manner, the correct outcome of user actions.

In particular, it is challenging to devise general purpose techniques that allow an application to balance the load imposed on servers by an increasing number of clients, and to enforce adequate semantics over the data accessed and manipulated by clients. To give a realistic example of such an application lets re-think the previous multi-user pacman online game. The application requires that all clients must, at all times, be able to read and write data from and to the server, and that all clients must constantly be updated by any changes that can have an impact on the users' actions. In practice, there are two main issues to think about when building such a system: first, networking wise, keeping all clients connected to the server farm and the aggregated required bandwidth; second, data wise, keeping data consistent and fresh at all time, led by the global write load imposed by all clients (each continuously updating data).

Proposal for a cloud-edge hybrid. The ultimate goal of this thesis is the design, implementation, and evaluation of mechanisms to mitigate the previously described issues. We intend to bridge the gap between the client and server side on two fronts.

First we bring the centralized service closer to the client. Instead of having multiple heavy datacenters managing the centralized logic of the application, we aim at dynamically partitioning logic and associated state of the application and migrate these to locations closer to the end user. We expect to concurrently make use of multiple cloud services (such as AWS, Azure, etc..) to be able to provision new instances as close as possible to end-users, improving user experience in terms of latency and performance.

This mostly differs from existing frameworks in the fact that they require manual addition of new datacenter locations. We envision a system where this can be achieved in a mostly autonomic fashion taking into consideration the execution environment and workloads generated by clients. This requires creating mechanisms to discover how to partition the data and how to dynamically add new partitions without compromising existing connections and performance at runtime.

Second, we intend to bring most of the application logic to the client side. Legion is well suited for collaborative applications, but applications like games, though possible to create, do not cope well with misbehaving users (i.e., users that lie regarding their view and history of operations to their own benefit). This can be addressed by creating mechanisms which are divided into two complementary aspects: adding stronger forms of consistency instead of only causal in our system; creating protocols that let applications be run in a peer-to-peer setting while preventing incorrect user behaviour.

Other forms of consistency are required for many use-cases, double spending being a concrete example. If users do not coordinate, in a game-like application with a restricted amount of resources each could spend all resources concurrently, leading to erroneous behaviour on the application after they communicate their operations with each other.

The system has to provide, besides a useful API to operate on data, clear ways to specify expected system behaviour. Not just on required consistency, on how to enable disconnected operation from the centralized component correctly, and on how data divergence due to concurrent operations is addressed, but also on how to restrict user actions to what is correct within the application. This is on viewing and/or editing data without the required permissions or editing data in an incorrect manner (i.e., what in games typically is seen as cheating, such as running $gold = gold * 2$ in the user console).

Notice that cheating is not restricted to where a user edits application data in an incorrect manner. Besides the requirement of only letting users edit data in a correct way in the eyes of an application, users must also be unable to tamper with the overall order of events (for example, suddenly claiming to use a health potion before dying).

Figure 3 depicts the envisioned system, where different cloud-services are used together and opportunistic peer-to-peer connections between clients are created.

Different applications have different requirements. Our approach to mitigate the fact of over compensating for some applications and not generalizing to any others at all is to create various application prototypes for evaluating our system. Creating these prototypes surely will enhance the results from this work and broaden many research aspects, as different applications have very different requirements. Furthermore, it enables us to evaluate our system in a

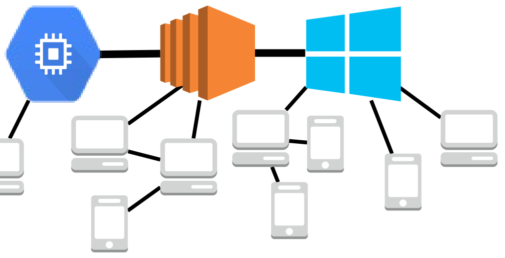

Figure 3: Diagram of the proposed System. Server instances on top (from left to right: Google Cloud Compute, Amazon EC2, Azure) and client devices below.

realistic setting, especially if the same applications are implemented in a baseline (client-server) system.

1.2 Expected Contributions

We expect to create a framework which allows for low effort development of large-scale user-centric applications. We expect results of this thesis to include:

- a distributed storage system supporting partial replication to address the needs of data sharing in massive-scale applications, which includes dealing with dynamic replica placement and migration;
- a communication middleware and supporting distributed algorithms for efficient communication among a large number of mobile devices and the centralized infrastructure;
- techniques for making direct (peer-to-peer) interaction among web and mobile users secure (privacy, tamper, and cheat-proof);
- providing the programming abstractions and algorithms for secure communications and mechanisms for detecting and mitigating incorrect behaviour;
- the platform which combines these techniques to simplify the development, deployment, and management of novel massive-scale web and mobile applications.

2 RELATED WORK

Our work has been influenced by prior research in multiple areas.

Internet services often run in cloud infrastructures composed by multiple data centers, and rely on a geo-replicated storage system [4–11] to store application data.

Some of these storage systems provide variants of weak consistency, such as eventual consistency [4] and causal consistency [5, 6, 9–11], where different clients can update different replicas concurrently without coordination. To ensure causal consistency we must adapt to a setting where we have a very large number of replicas writing on the data (clients locally, and propagating in a peer-to-peer fashion), which these previous works do not address.

Other storage systems adopt stronger consistency models, such as parallel snapshot isolation [12] and linearizability [8], where concurrent (conflicting) updates are not allowed without some form of coordination. In our context the algorithms used to coordinate access to data storage for executing each update are prohibitively expensive for high throughput and large numbers of clients (manipulating the same set of data objects).

Replication at the clients has been proposed in the past. In the context of mobile computing [13], systems such as Coda [14] and Rover [15] support disconnected operation relying on weak consistency models. Parse [16], SwiftCloud [17] and Simba [18] are recent systems that allow applications to access and modify data during periods of disconnection. While Parse provides only an eventual consistency model, SwiftCloud additionally supports highly available transactions [19] and enforces causality. Simba allows applications to select the level of observed consistency: eventual, causal, or serializability. In contrast to these systems, we intend clients to synchronize directly with each other, reducing latency of update propagation and allowing collaboration when disconnected from servers.

Collaborative applications and frameworks maintain replicas of shared data in client machines. Etherpad [20] allows clients to collaboratively edit documents. ShareJS [21] and Google Drive Realtime [22] are generic frameworks that manage data sharing among multiple clients. All these systems use a centralized infrastructure to mediate interactions among clients as they rely on operational transformation to guarantee eventual convergence of replicas [23, 24], where we have as goal not to rely on the constant connection to a server.

Peer-to-Peer systems and Fog Computing Extensive research on decentralized overlay networks [25–27] and gossip-based multicast protocols [25, 28, 29] has been produced in the past. Although our design for supporting peer-to-peer communication among clients builds on previous designs, so far it mostly differs in the way we promote low latency links among clients and leverage the centralized infrastructure.

Fog Computing [30], is a close topic to our research, but so far most research approaches Internet of Things as their use case (i.e., networks of sensors and actuators, often wireless). Nevertheless, many aspects of Fog Computing apply to our work, and research in these topics must be taken into account, such as heterogeneity of devices, geographical distribution of nodes, security related aspects and, if supporting mobile, battery management.

Security The use of Trusted Execution Environment (TEE) like Intel SGX has been argued for securing games, one example being [31]. These works focus on protecting the integrity and confidentiality of code and data of a single-player game (i.e., DRM), whereas we focus on multi-player games where players themselves interact in a peer-to-peer fashion.

TrustJS [32] is very interesting to our use-cases as it allows for JavaScript to run, in a secure-fashion, at the client side. In out work we do not want to restrict an application to require the existence of a TEE, but to only use it when needed and if available at the client side (resorting to another trusted component, like the server, otherwise). For example, in a simple game like tic-tac-toe, each client should be able to trivially verify operations received from other users. In cases where a trusted component is indeed necessary, we want to be able to chose from various options depending on what is available and acceptable: use a TEE existing at one of the players, use an available TEE existing at any other peer, or using a (trusted) server if all else fails or when it is simply the only acceptable choice latency wise.

Albert van der Linde
a.linde@campus.fct.unl.pt
NOVA LINCS & DI-FCT-UNL

REFERENCES

[1] Albert van der Linde, Pedro Fouto, João Leitão, Nuno Preguiça, Santiago Castiñeira, and Annette Bieniusa. Legion: Enriching internet services with peer-to-peer interactions. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17.

[2] Anna Bouch, Allan Kuchinsky, and Nina Bhatti. Quality is in the eye of the beholder: Meeting users' requirements for internet quality of service. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, pages 297–304, New York, NY, USA, 2000. ACM.

[3] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The effects of loss and latency on user performance in unreal tournament 2003®. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '04, pages 144–151, New York, NY, USA, 2004. ACM.

[4] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6), 2007.

[5] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. In *Proc. of SOSP'11*, 2011.

[6] Sérgio Almeida, João Leitão, and Luís Rodrigues. Chainreaction: A causal+ consistent datastore based on chain replication. In *Proc. of EuroSys'13*, 2013.

[7] Brian F Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. of VLDB Endow.*, 1(2), 2008.

[8] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM TOCS*, 31(3), 2013.

[9] Deepthi Devaki Akkoorath, Alejandro Z Tomsic, Manuel Bravo, Zhongmiao Li, Tyler Crain, Annette Bieniusa, Nuno Preguiça, and Marc Shapiro. Cure: Strong semantics meets high availability and low latency. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 405–414. IEEE, 2016.

[10] Syed Akbar Mehdi, Cody Littley, Natacha Crooks, Lorenzo Alvisi, Nathan Bronson, and Wyatt Lloyd. I can't believe it's not causal! scalable causal consistency with no slowdown cascades. In *NSDI*, pages 453–468, 2017.

[11] Manuel Bravo, Luís Rodrigues, and Peter Van Roy. Saturn: a distributed metadata service for causal consistency. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 111–126. ACM, 2017.

[12] Yair Sovran, Russell Power, Marcos K. Aguilera, and Jinyang Li. Transactional storage for geo-replicated systems. In *Proc. of SOSP'11*, 2011.

[13] Douglas B. Terry. *Replicated Data Management for Mobile Computing*. Synthesis Lectures on Mobile and Pervasive Computing. Morgan & Claypool Publishers, 2008.

[14] James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM TOCS*, 10(1), February 1992.

[15] A. D. Joseph, A. F. de Lespinasse, J. A. Tauber, D. K. Gifford, and M. F. Kaashoek. Rover: A toolkit for mobile information access. In *Proc. SOSP'95*, 1995.

[16] Parse. parse.com.

[17] Marek Zawirski, Nuno Preguiça, Sérgio Duarte, Annette Bieniusa, Valter Balegas, and Marc Shapiro. Write Fast, Read in the Past: Causal Consistency for Client-side Applications. In *Proc. of Middleware'15*. ACM/IFIP/Usenix, December 2015.

[18] Dorian Perkins, Nitin Agrawal, Akshat Aranya, Curtis Yu, Younghwan Go, Harsha V. Madhyastha, and Cristian Ungureanu. Simba: Tunable End-to-end Data Consistency for Mobile Apps. In *Proc. of EuroSys '15*, 2015.

[19] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly available transactions: Virtues and limitations. *Proc. of VLDB Endow.*, 7(3), November 2013.

[20] EtherpadFoundation. Etherpad. etherpad.org.

[21] Joseph Gentle. ShareJS API. github.com/share/ShareJS.

[22] Google Inc. Google Drive Realtime API. developers.google.com/google-apps/realtime/overview.

[23] David A Nichols, Pavel Curtis, Michael Dixon, and John Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proc. UIST'95*, 1995.

[24] Chengzheng Sun and Clarence Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proc. of Comp. supported cooperative work*, 1998.

[25] João Leitão, José Pereira, and Luis Rodrigues. Hyparview: A membership protocol for reliable gossip-based broadcast. In *Proc. of DSN'07*. IEEE, 2007.

[26] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. of Net. & Sys. Manag.*, 13(2), 2005.

[27] Ayalvadi Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In *Net. Group Comm.* 2001.

[28] Kenneth P Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM TOCS*, 17(2), 1999.

[29] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues. Emergent structure in unstructured epidemic multicast. In *Proc. of DSN'07*, UK, June 2007.

[30] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[31] Erick Bauman and Zhiqiang Lin. A case for protecting computer games with sgx. In *Proceedings of the 1st Workshop on System Software for Trusted Execution*, SysTEX '16, pages 4:1–4:6, New York, NY, USA, 2016. ACM.

[32] David Goltzsche, Colin Wulf, Divya Muthukumaran, Konrad Rieck, Peter Pietzuch, and Rüdiger Kapitza. Trustjs: Trusted client-side execution of javascript. In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17, New York, NY, USA, 2017. ACM.