Building Cloud Applications for Challenged Networks

Yehia Elkhatib(✉)

School of Computing and Communications, Lancaster University, Lancaster, UK
y.elkhatib@lancaster.ac.uk

Abstract. Cloud computing has seen vast advancements and uptake in many parts of the world. However, many of the design patterns and deployment models are not very suitable for locations with challenged networks such as countries with no nearby datacenters. This paper describes the problem and discusses the options available for such locations, focusing specifically on community clouds as a short-term solution. The paper highlights the impact of recent trends in the development of cloud applications and how changing these could better help deployment in challenged networks. The paper also outlines the consequent challenges in bridging different cloud deployments, also known as *cross-cloud computing.*

1 Introduction

Cloud computing is yet another step on the quest for holy grail of computing as described by John MacCarthy whilst addressing the MIT Centennial in 1961:

> *If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry.*

Indeed, the cloud computing market is a vast and rapidly expanding one, worth $58bn in 2013 and expected to grow to $191bn by 2020 [19]. It offers its users unparallaled flexibility and scalability, allowing them to easily and feasibly scale a system up and down to meet changing business targets (e.g. customer demand), to dynamically mitigate system failures through the spawning of new servers, and to expeditiously and seamlessly roll out new capabilities. This allows businesses to curb computing expenses whilst still supporting agile business progress. For researchers, it imposes much lower resource provisioning barriers that the closest alternatives (HPC or grid): virtually immediate resource provisioning, huge scale to store and process vast observational datasets, and less restrictions on location. This enables researchers using the cloud to focus on their processes rather than on the computing infrastructure, and to run computations and access data when and where they require.

R. Horne (Ed.): EGC 2015, CCIS 514, pp. 1–10, 2015.
DOI: 10.1007/978-3-319-25043-4_1

2 Problem Space

First, one needs to define the term *emerging economies* in order to add some clarity, albeit in the boundaries of this work, to such usually loosely defined term. An emerging economy (EE) is one with a relatively stable government, growing investment in public infrastructure, and a prospering consumer market typically manifested by an expanding middle class with increasing levels of disposable income. Under such definition, example os EEs include countries like Kazakhstan, Malaysia and Turkey.

In EEs, access to resources is not the main obstacle in the way to encourage uptake of the cloud. On the contrary, both infrastructure provisions and end user devices are fairly affordable for a large fraction of the local population. The main problem is to do with the network connection. This is a problem because currently all applications are distributed [7]. This is partly due to an ever increasing reliance on communication between applications to carry out transactions, and on interaction between users for social and productivity purposes. This is also due to a shift in application provisioning: most applications are now hosted remotely and delivered remotely to the end users. Gone are the days of shrink-wrapped software.

Such shift introduces an increasing expectation on good network performance, which many locations do not have. In this paper, I focus on such locations with challenged network connections.

3 Latency Is High, So What?

Making more bandwidth is easy [8], which is why Internet Service Providers (ISPs) all around the world will always market their services based on bandwidth. However, having more bandwidth does not always translate to better network performance [4]. On the other hand, decreasing network latency is much more effective in increasing application throughput [12]. In this regard, significant efforts in the network research community specifically target the issue of inflated network latency. A recent example of this is SPDY [2], a web protocol that multiplexes HTTP connections to reduce the number of round trips between client and server. This, however, is only effective in reducing the bursty nature of HTTP for certain network connections [10].

High network latency has an obvious detrimental effect on latency-sensitive applications. In the gaming industry, 100 ms is considered to be the threshold between an interactive and unresponsive experience [9,17]. For Voice over IP (VoIP), the threshold is between 100 ms [27] and 150 ms [15] for acceptable audio transmission. On e-commerce websites, increments of just 100 ms can decrease sales by 1 % [18]. Similarly, it is reported that most web users would not tolerate beyond 2 s "for simple information retrieval tasks on the Web" [22], which demands significantly low latency between the client and the different servers serving different pieces of content on a typical webpage.

Even bandwidth-hungry applications, such as video streaming, can be adversely affected by relatively high delay levels. Tuning TCP send and receive buffers ensures that the amount of data in transit at any time is the maximum that the link between the sender and receiver could accommodate. Untuned TCP buffers fails to attain the maximum achievable throughput. Despite the knowledge of this fact in the networking community for many years (cf. [20]), manually optimising TCP buffers requires good technical nous [11]. Various efforts over the years attempt to move away from the black art of manual TCP tuning to automatically adjusted TCP buffer sizes and other networking parameters based on connection characteristics [28,33,34]. Nonetheless, such solutions are still not included by default in modern operating systems.

4 Design Principles

In this section, I revisit some of the fundamental design principles of modern day distributed systems in light of the constraints imposed by challenged networks. I propose alternative design choices for each principle, giving examples where possible.

4.1 Application Signalling

One of the unintentional products of the easy integration facilitated by cloud computing services is a move towards building *chatty applications*. These are applications that interact with each other at a high network cost manifested as a very high number of messages and/or large message sizes.

Such verbosity is partly due to a high skewness in the cloud application space. I conjecture that this a by-product of the startup culture, where most applications are built in order to be sold to a much larger company or get funds from a venture capitalist. Under such conditions services are engineered to demonstrate high impact and revenue. Hence, the majority of applications emanating from the silicon valleys of the world are developed for the highly connected metropolises in the industrialised world where they originate; e.g. San Francisco, New York, London, etc.

Hidden within this context is an obvious assumption of relative proximity to industrial-scale data centers. At the same time, cloud service providers (CSPs) over provision their services to be able to handle a thick and fast stream of incoming API calls. Hence, the only cost for verbose API interactions is monetary. The cost models of major CSPs make this relatively affordable for most applications anyway. This is the root of the recent emergence of chatty applications. Loose-coupling is only really considered when it starts to be too expensive to shift large data sets or when it becomes difficult (or impossible) to handle large volumes of updates.

However, this is not suitable for EEs with relatively high RTT where network communication has to be rationed. Therefore, it is important to design applications that can cope with high latency, i.e. have low signalling or "less chatty" as a design requirement. In this light, the communication style of the application is expected to lean more towards asynchronicity. For example, 'fire and forget' asynchronous remote procedure calls (RPC) is much more suitable than the currently common trend of blocking whilst an API call is completed. Similarly, message oriented middleware (MOM) solutions offer loosely-coupled communications with little time guarantees.

4.2 Data Exchange

One result of the assumption of proximity to data centres is diminishing the third tier in the classical 3-tier distributed systems model: the data tier. Instead, a lot of the data is constantly being sent over the network back and forth between machines and clients as and when required. The data tier, typically a relational or a NoSQL database, is only used for huge bulks of data. This trend if facilitated by the Representational State Transfer (REST) architectural design principles [13]. Indeed, it suits the cloud as we now know it: Web services are completely stateless and hence have less load on them. Clients invoke services with the data required to transition between different states. Virtual machines (VMs) hosting the web services are easily replaceable and replicatable. This vastly improves scalability and simplifies infrastructure management tasks such as load balancing and failure recovery.

For EEs, the data tier needs to be reinstated at least for deployments in challenged network conditions. This is manifested as separate replicas that are more geared towards the users it serves, perhaps based on geographical location. Restricting write updates based on domain is an obvious policy to minimise write-write conflicts between different replicas. Lazy updates with remote replicas would be sufficient for most applications, however this is discussed in more detail in the following subsection. Such setup still preserves the ability to create large application platforms that are made up of stateless VMs that are easy to spin up or stop as and when required, whilst also maintaining a certain degree of independence on data exchange over WAN links of relatively low performance levels.

4.3 Data Repositories

The famous Brewer's (or CAP) theorem states that a database system can attain a maximum of two of the following three objectives: **c**onsistency, **a**vailability, and **p**artition tolerance [5]. In traditional distributed database systems, consistency and partition tolerance were valued more than availability. In contrast, the stateless nature of many cloud applications forsakes consistency, and instead prioritises availability and partition tolerance. We see this reflected in the popularity of many 'eventually consistent' NoSQL systems such as Cassandra, Couchbase, Dynamo, MongoDB, Riak, and more.

For applications in network challenged areas, partition tolerance has to be the highest priority. This is to allow the system to continue operation if network connectivity is lost or is of poor quality. The trade-off, then, becomes between availability and consistency. This is dependent on the application in hand. For some applications, such as social networks and multimedia systems, availability is perhaps more important than consistency with the rest of the remote system. On the other hand, applications such as e-commerce and customer relationship management would place consistency as a higher priority as asynchronous transaction could cost more than a missed one. As such, data repositories need to be engineered in such a way to cater for both the needs of the application and the restrictions imposed by the deployment environment.

4.4 Communication Protocols

The TCP/IP stack has become an entrenched part of most distributed systems we have today. On top of this, HTTP and HTTPS have also become a de facto communication standards for the majority of cloud applications and APIs [25]. However, this protocol stack needs to be reconsidered to confirm suitability for the target deployment environment, alongside TCP optimisations discussed in Sect. 3. Note that networking stack used to interconnect system elements could be different from that enabling user access.

For instance, SPDY (and more recently HTTP/2.0) offer better performance by multiplexing different data streams between two hosts (such as a client and a server) onto one TCP connection. This saves significant amount of unnecessary connection establishment time, which brings considerable improvements especially over network links with high latency [10]. However, it is also susceptible to performance degradation on links with relatively high loss rate [ibid].

The IP protocol is the default choice for traditional host address-oriented network architectures. However, alternative network addressing paradigms could offer alternative solutions that might be more suitable for challenged networks. An example of this is information-centric networking (ICN) [16,31]. To date, only one large-scale simulation study is available as evidence of ICN's promise [30] but is focused on caching content in a peer-to-peer information exchange network. Studies tailored specifically for the conditions of challenged networks would be very useful.

5 Discussion

In this section, I discuss two main approaches to solving the challenges highlighted in the paper thus far.

5.1 Top-Down Approach

An answer to the challenges to distributed systems in EEs as highlighted above is to build local or regional data centers. This is indeed a comprehensive solution that is worth investigating, especially with the special context of many EEs that dictates possible innovations such as using alternative energy sources. However, such solution involves huge long-term projects that require large budgets, significant depth and range of expertise, and long-term geopolitical planning. In other words, it will not bring instant solutions to people needing to access cloud resources in the short term.

5.2 Bottom-Up Approach

An alternative strategy is to provide relatively low-cost systems that are rapidly realisable, even if they only offer partial solutions. The concept of community clouds offers a rewarding option in this regard.

Community clouds is a cooperative model for deploying clouds driven by certain restrictions. In the context discussed here, the constraint is limited connectivity. The model relies on using free open source software solutions and affordable hardware to provide a self-service infrastructure. It is similar in spirit to the cooperative models of wireless mesh networks and volunteer computing (e.g. SETI@Home, BOINC). A typical setup would use OpenStack over a small set of commodity hardware machines. This brings relatively rapid access to cloud resources as well as active involvement in assembly and operation.

Examples of innovative community cloud solutions are starting to emerge. Of these, I present only three for illustration purposes.

Cloud&Heat Technologies[1] [1] is a German company offering mini-clouds that recycle the heat they emitted. A mini-cloud is a set of self-contained fire-proof cabinets that are installed in the basements of residential and commercial buildings. They comprise of between one and six cabinets that operate as a single OpenStack deployment connected by broadband to the wider Internet. The deployment is capable of operating a range of cloud appliances obtained via the Bitnami open source marketplace. The heat produced by the cabinets is used to warm a water buffer tank for domestic purposes (e.g. washing) and central heating radiators.

Another example linked to energy efficiency in domestic environments comes from Qarnot[2]. This French company designed a domestic central heating radiator module that is in fact a multi-processor HPC cluster. The objective is to attract intensive computing jobs to locations where heat is needed. When the user turns up the thermostat, enough extra computation is directed from corporate clients to increase the emitted heat at the user's location. Additional electricity costs required for carrying out the computation are refunded to the user. The Qarnot cluster supports different job types of scientific applications, such as AutoDock, Blender, Gromacs, NAMD, NWChem, OpenFOAM, Python, Quantum Espresso, and R.

A third example targets ad hoc cloud deployments in order to make them reliable enough for executing demanding applications. SCADAMAR [6] uses different network overlays to run MapReduce jobs over volunteer clouds. The Berkeley Open Infrastructure for Network Computing (BOINC) [3] is used to access underused resources on different devices in the volunteer cloud. BitTorrent is used as a substrate to distribute input, intermediate and output data between the nodes in the system (mappers and reducers). A bespoke scheduler is used to tolerate node failures, enabling the system to be resilient and to achieve job throughput over a potentially high number of nodes.

[1] http://www.cloudandheat.com/.

[2] http://www.qarnot-computing.com/.

6 Cross-Cloud Computing

The subsequent challenge is to piece the different cloud blocks together to form a larger pool of resources without barriers and with minimum gravity to any of the blocks in particular. In other words, solutions must be developed in order to enable applications to easily straddle different cloud infrastructures in a hybrid fashion, and to ease difficulties of moving between different imputing infrastructures. I refer to this challenge as 'cross-cloud computing'.

The difficulty of cross-cloud computing arises from a number of sources. There is the classical interoperability problem, manifested here in the form of APIs that are divergent both semantically and syntactically. There is also a significant obstacle in porting applications between cloud infrastructures due to the different formats used to capture the state of running and idle VMs, as well as due to the high network overhead associated with such migration practices. In the following, I give an overview of the migration options as they currently stand and the possible improvements that could be delivered to alleviate some of the problems in this area.

6.1 Entry

Developers wishing to enter the cloud market are faced with an array of questions that are in some cases quite difficult to answer: *(a)*Which provider? *(b)* Which instance type(s)? *(c)* Which availability zone? *(d)* What time? This is complicated with an overwhelmingly large range of options for each question. Moreover, the answers to these questions are quite subjective but would greatly determine the incurred cost and performance of the application. Hence, work is required to aid developers decide the best deployment environment for their application and to re-evaluate such decisions as the application requirements and the cloud market offerings change. Some work is already starting to happen in this area, e.g. [23,26,32].

The concern of divergent APIs could be sufficiently addressed for many use cases by employing any of the number of multi-cloud common programming models or libraries available for such purpose. These include jclouds[3], Brooklyn[4], Scalr[5], SeaClouds [24], and others. They provide a 'least common denominator' between the different APIs which, as already mentioned, is sufficient for many cloud application developers whose intent does not exceed starting and stopping instances. However, any need outside the least common denominator API, e.g. billing, is not supported and needs to be addressed individually by the developer. This obviously detracts from the value of such solutions.

[3] http://jclouds.apache.org/.
[4] https://brooklyn.incubator.apache.org/.
[5] http://www.scalr.com/.

6.2 Migration

To port a workload between different cloud infrastructures, the options are currently as follows. One could package memory and disk state along with associated metadata onto a virtual disk image and transfer that across. This is the most network expensive method as such 'pre-baked VM images' could be a few gigabytes in size. In addition, different cloud providers accept different sets of machine image formats.

An alternative is to use configuration management tools (CMTs), such as Chef[6] and Ansible[7]. These allow a developer to express in code what their application needs in terms of number of instances and the software that needs to be set on each instance. Despite quite promising in theory, they are not as deterministic as one would want them to be as there are non-reconcilable differences in the results achieved over varying operating systems [35]. Another hidden cost is developer time: CMTs offer no error diagnosis or remediation support if the desired execution environment is not produced. Instead, the developer has to revert to manual modifications.

A third approach is to replace reliance on VMs by employing containers as lightweight isolated execution environments. Despite being established in the Linux community for years (cf. [29]), containers as a concept have gained significant attention in recent months due to the rise of technologies such as Docker [21] and Rocket[8]. These provide developers with very accessible and controlled means of packaging and distributing software over lightweight OS-supported containers. This is indeed very useful for testing and rolling out new services. However, they are designed for immutable appliances and as such are not suitable for migrating stateful appliances that are in operation. An alternative is MultiBox [14] which utilises only Linux-native features to support minimalist containers to decouple guest processes from the host machine. This offers a migration vehicle that is completely independent of the CSP as long as they provide Linux VMs on top of which MultiBox could operate.

7 Conclusion

Cloud computing offers great potential for building elastic and agile applications in different sectors. There are additional challenges to capitalising on this potential for those wanting to deploy cloud applications under challenged network conditions. In this paper, I focused specifically on the case of emerging economies. I highlighted network latency as the main problem with such environments, and I discussed how the design and implementation of cloud applications need to be consequently changed. I then presented an overview of community clouds as a feasible short-term solution for founding cloud infrastructures in a grassroots fashion. Finally, I identified cross-cloud computing as the key future challenge in

[6] https://www.chef.io/.
[7] http://www.ansible.com/.
[8] https://coreos.com/blog/rocket/.

terms of breaking down the barriers between such grassroots deployments, and also to open then up to the rest of the global cloud ecosystem.

Acknowledgments. The author is grateful for the organisers of the International Workshop on Embracing Global Computing in Emerging Economies, and in particular Dr. Ross Horne, for their invitation to present and discuss an earlier version of this work. This work was supported in part by the CHIST-ERA Dionasys project grant reference EP/M015734/1.

References

1. Cloud&Heat - the efficient cloud service. http://www.cloudandheat.com/
2. SPDY: An experimental protocol for a faster web. http://www.chromium.org/spdy/spdy-whitepaper
3. Anderson, D.P.: BOINC: a system for public-resource computing and storage. In: Fifth IEEE/ACM International Workshop on Grid Computing, pp. 4–10, November 2004
4. Belshe, M.: More bandwidth doesn't matter (much). Google Inc. (2010)
5. Brewer, E.: CAP twelve years later: how the "rules" have changed. Computer **45**(2), 23–29 (2012)
6. Bruno, R., Ferreira, P.: SCADAMAR: Scalable and data-efficient internet mapreduce. In: Proceedings of the CrossCloud Brokers International Workshop, pp. 2:1–2:6. ACM, December 2014
7. Cavage, M.: There's just no getting around it: You're building a distributed system. Queue **11**(4), 30–41 (2013)
8. Cheshire, S.: It's the latency, stupid, May 1996. http://www.stuartcheshire.org/rants/Latency.html
9. Claypool, M., Claypool, K.: Latency and player actions in online games. Commun. ACM **49**(11), 40–45 (2006)
10. Elkhatib, Y., Tyson, G., Welzl, M.: Can SPDY really make the web faster?. In: Proceedings of IFIP International Conference on Networking, June 2014
11. Mahdavi, J., et al.: Enabling high performance data transfers. http://www.psc.edu/index.php/networking/641-tcp-tune
12. Fall, K., McCanne, S.: You don't know jack about network performance. Queue **3**(4), 54–59 (2005)
13. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis. University of California, Irvine (2000)
14. Hadley, J., Elkhatib, Y., Blair, G.S., Roedig, U.: Multibox: lightweight containers for vendor-independent multi-cloud deployments. In: Horne, R. (ed.): EGC 2015, CCIS 514, pp. 1–12 (2015)
15. International Telecommunication Union. Recommendation G.114: One-way transmission time, May 2003
16. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H., Braynard, R.L.: Networking named content. In: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT 2009, pp. 1–12. ACM (2009)
17. Jarschel, M., Schlosser, D., Scheuring, S., Hossfeld, T.: An evaluation of QoE in cloud gaming based on subjective tests. In: International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 330–335, June 2011

18. Kohavi, R., Longbotham, R.: Online experiments: lessons learned. Computer **40**(9), 103–105 (2007)
19. KPMG. Cloud survey report: Elevating business in the cloud, October 2014. http://www.kpmginfo.com/EnablingBusinessInTheCloud/downloads/2014%20KPMG%20Cloud%20Survey%20Report%20-%20Final%2012-10-14.pdf
20. Lakshman, T., Madhow, U.: The performance of TCP/IP for networks with high bandwidth-delay products and random loss. IEEE/ACM Trans. Netw. **5**(3), 336–350 (1997)
21. Merkel, D.: Docker: lightweight linux containers for consistent development and deployment. Linux J. **2014**(239), 2 (2014)
22. Nah, F.F.-H.: A study on tolerable waiting time: how long are web users willing to wait? Behav. Inf. Technol. **23**(3), 153–163 (2004)
23. Papakos, P., Capra, L., Rosenblum, D.S.: VOLARE: Context-aware adaptive cloud service discovery for mobile systems. In: Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware, ARM 2010, pp. 32–38. ACM (2010)
24. Petcu, D., Di Nitto, E., Ardagna, D., Solberg, A., Casale, G.: Towards multi-clouds engineering. In: 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), April 2014
25. Popa, L., Ghodsi, A., Stoica, I.: HTTP as the narrow waist of the future internet. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, pp. 6:1–6:6. ACM (2010)
26. Samreen, F., Blair, G.S., Rowe, M.: Adaptive decision making in multi-cloud management. In: Proceedings of the CrossCloud Brokers International Workshop, pp. 4:1–4:6. ACM, December 2014
27. Sat, B., Wah, B.W.: Analyzing voice quality in popular VoIP applications. IEEE Multimed. **16**(1), 46–59 (2009)
28. Semke, J., Mahdavi, J., Mathis, M.: Automatic TCP buffer tuning. ACM SIGCOMM Comput. Commun. Rev. **28**(4), 315–323 (1998)
29. Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.: Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. ACM SIGOPS Operating Syst. Rev. **41**(3), 275–287 (2007)
30. Tyson, G., Kaune, S., Miles, S., Elkhatib, Y., Mauthe, A., Taweel, A.: A trace-driven analysis of caching in content-centric networks. In: Proceedings of the 21st International Conference on Computer Communications and Networks (ICCCN 2012). IEEE, August 2012
31. Tyson, G., Sastry, N., Rimac, I., Cuevas, R., Mauthe, A.: A survey of mobility in information-centric networks: Challenges and research directions. In: Proceedings of the 1st ACM Workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications, NoM 2012, pp. 1–6. ACM (2012)
32. Vanbrabant, B., Joosen, W.: Configuration management as a multi-cloud enabler. In: Proceedings of the CrossCloud Brokers International Workshop, pp. 1:1–1:3. ACM, December 2014
33. Weigle, E., chun Feng, W.: A comparison of TCP automatic tuning techniques for distributed computing. In: Proceedings of the IEEE HPDC, pp. 265–272 (2002)
34. Winstein, K., Balakrishnan, H.: TCP ex machina: Computer-generated congestion control. ACM SIGCOMM Comput. Commun. Rev. **43**(4), 123–134 (2013)
35. Zhu, L., Xu, D., Xu, X.S., Tran, A.B., Weber, I., Bass, L.: Challenges in practicing high frequency releases in cloud environments. In: Secnd International Workshop on Release Engineering, Mountain View, USA, pp. 21–24, April 2014