Non-functional Requirements for Real World Big Data Systems
An Investigation of Big Data Architectures at Facebook, Twitter and Netflix

Thalita Vergilio and Muthu Ramachandran
School of Computing, Creative Technologies and Engineering, Leeds Beckett University, Leeds, U.K.

Keywords: Big Data, Facebook, Twitter, Netflix, Scalability, Distribution, Fault Tolerance, Processing Guarantees.

Abstract: This research represents a unique contribution to the field of Software Engineering for Big Data in the form of an investigation of the big data architectures of three well-known real-world companies: Facebook, Twitter and Netflix. The purpose of this investigation is to gather significant non-functional requirements for real-world big data systems, with an aim to addressing these requirements in the design of our own unique architecture for big data processing in the cloud: MC-BDP (Multi-Cloud Big Data Processing). MC-BDP represents an evolution of the PaaS-BDP architectural pattern, previously developed by the authors. However, its presentation is not within the scope of this paper. The scope of this comparative study is limited to the examination of academic papers, technical blogs, presentations, source code and documentation officially published by the companies under investigation. Ten non-functional requirements are identified and discussed in the context of these companies' architectures: batch data, stream data, late and out-of-order data, processing guarantees, integration and extensibility, distribution and scalability, cloud support and elasticity, fault-tolerance, flow control, and flexibility and technology agnosticism. They are followed by the conclusion and considerations for future work.

1 INTRODUCTION

Big data is defined as data that challenges existing technology for being too large in volume, too fast, or too varied in structure. Big data is also characterised by its complexity, with associated issues and problems that challenge current data science processes and methods (Cao, 2017). Large internet-based companies have the biggest and most complex data, which explains their leading role in the development of state-of-the-art big data technology.

This paper contributes to the existing knowledge in the area of Software Engineering for Big Data by performing a search of the existing literature published by three major companies, and an outline of the strategies devised by them to cope with the technological challenges posed by big data in their production systems. Non-Functional requirements are important quality attributes which influence the architectural design of a system (Chung and Prado Leite, 2009). Ten non-functional requirements for big data systems are identified and discussed in the context of these real-world implementations. These requirements are used to guide the design and development of a new architecture for big data processing in the cloud: MC-BDP. The presentation, evaluation and discussion of MC-BDP shall be addressed in a future paper.

The companies targeted for this study are Facebook, Twitter and Netflix. The methodology used in this comparative study is explained in Section 2, which covers the scope of this research, as well as the selection criteria used. Section 4 presents the non-functional requirements and discusses how they are implemented by the three companies in their production systems. Finally, section 5 presents the conclusion and considerations for future work.

2 METHODOLOGY

This section starts by defining the scope of this research. This is followed by an explanation of the criteria used to select the companies under examination.

2.1 Scope

The scope of this paper is limited to academic papers, technical documentation, and presentations or blog posts officially published by the companies under evaluation. Table 1 shows a summary of the materials used as source for this research, classified by type.

Table 1: Classification and summary of source materials.

Company	Academic Paper	Technical Blog	Presentation	Code/ Documentation
Facebook	2	2	1	1
Twitter	2	2	1	3
Netflix	1	8	2	0

2.2 Selection Criteria

An initial survey was conducted, limited to peer-reviewed academic papers. Three search engines were primarily used to perform the searches: Google Scholar, IEEE Xplore Digital Library and ACM Digital Library. The initial survey searched for terms such as “big data”, “big data processing”, “big data software” and “big data architecture”. For the sake of thoroughness, synonyms were used to replace key terms where appropriate, e.g. “system” for “software”.

The first classification which became apparent was in terms of who developed the solutions presented. The results found comprised technologies developed **1)** by academia, **2)** by real-world big data companies, **3)** by industry experts as open-source projects, or **4)** by a combination of the above. This research focuses on category number 2.

A further classification can be drawn from the academic papers reviewed, this time in terms of how the contributions presented were evaluated. Three cases were encountered:

A) cases where there is no empirical evaluation of the proposed solution.

B) cases where the empirical evaluation of the proposed solution is purely experimental.

C) cases where peer-reviewed published material was found describing the results of implementing the proposed solution in large-scale commercial big data settings.

In order to select suitable companies to include in this study, the focus of this research was limited to category C.

Three companies or cases were selected within the criteria characterised above: Facebook, Twitter and Netflix. These were selected from a wider pool of qualifying companies which included Microsoft (Eliot, 2010), (Bernstein et al., 2014), Google (Akidau et al., 2013), (Akidau et al., 2015), and Santander (Cheng et al., 2015). The rationale for choosing the three aforementioned companies is based on the quantity, quality and clarity of the information encountered, as well as availability of technical material online such as project documentation and architectural diagrams.

3 NON-FUNCTIONAL REQUIREMENTS

This section presents ten non-functional requirements discussed in the literature published by the three companies selected in section 2.2. It then examines how they implemented these requirements and compares the different solutions.

3.1 Batch Data

This requirement refers to the capability to process data which is finite and usually large in volume, e.g. data archived in distributed file systems or databases.

Both Facebook and Twitter estimate that the finite data they hold on disk reaches hundreds of petabytes, with a daily processing volume of tens of petabytes (Krishnan, 2016). Netflix’s big data is one order of magnitude smaller, with tens of petabytes in store and daily reads of approximately 3 petabytes (Gianos and Weeks, 2016).

Facebook uses a combination of three independent, but communicating systems to manage its stored data: an Operational Data Store (ODS), Scuba, Hive and Laser (Chen et al., 2016).

Twitter’s batch data is stored in Hadoop clusters and traditional databases, and is processed using Scalding and Presto (Krishnan, 2016). Scalding is a Scala library developed in-house to facilitate the specification of map-reduce jobs (Twitter, Inc., 2018). Presto, on the other hand, was originally developed by Facebook. It was open-sourced in 2013 (Pearce, 2013), and has since been adopted not only by Twitter, but also by Netflix (Tse et al., 2014).

Differently from the previous two companies, Netflix’s Hadoop installation is cloud-based, and it uses an in-house developed system called Genie to manage query jobs submitted via Hadoop, Hive or Pig. Data is also persisted in Amazon S3 databases (Krishnan and Tse, 2013).

3.2 Stream Data

This requirement refers to the capability to process data which is potentially infinite and usually flowing at high velocity, e.g. monitoring data captured and processed in real-time, or close to real-time.

Stream processing at Facebook is done by a suite of in-house developed applications: Puma, Swift and Stylus. Puma is a stream processing application with a SQL-like query language optimised for compiled queries. Swift is a much simpler application, used for checkpointing. Finally, Stylus is a stream processing framework which combines stateful or stateless units of processing into more complex DAGs (Chen et al., 2016).

Storm, one of the most popular stream processing frameworks in use today, was developed by Twitter (Toshniwal et al., 2014). Less than five years after the initial release of Storm, however, Twitter announced that it had replaced it with a better performing system, Heron, and that Storm had been officially decommissioned (Fu et al., 2017). Heron uses Mesos, an open-source cluster management tool designed for large clusters. It also uses Aurora, a Mesos framework developed by Twitter to schedule jobs on a distributed cluster.

Netflix also uses Mesos to manage its large cluster of cloud resources. Scheduling is done by a custom library called Fenzo, whereas stream processing is done by Mantis, which is also custom-developed.

3.3 Late and out of Order Data

This requirement relates to stream processing and refers to the capability to process data which arrives late or in a different order from that in which it was emitted. Streaming data from mobile users, for example, could be delayed if the user loses reception for a moment. In order to handle late and out of order data, a system must have been designed with this requirement in mind.

All three streaming architectures utilise the concept of windows of data to transform infinite streaming data into finite windows that can be processed individually.

For handling late and out of order data, Facebook's Stylus utilises low watermarks. No mention was found in Twitter Heron's academic paper of whether it provides a mechanism for dealing with late or out of order data. However, looking at the source code for the Heron API, the BaseWindowedBolt class, merged into the master project in 2017, has a method called withLag(), which allows the developer to specify the maximum amount of time by which a record can be out of order (Peng, 2017).

No mention was found in documentation published by Netflix of Mantis's strategy for dealing with late and out of order data. Because the source code for Mantis is proprietary, further investigation was limited.

3.4 Processing Guarantees

This requirement refers to a stream system's capability to offer processing guarantees, i.e. exactly once, at least once and at most once. While exactly once processing is ideal, it comes at a cost which could translate into increased latency.

Exactly once semantics involves some level of checkpointing to persist state. There is therefore an inherent latency cost associated with it, which is why not all use-cases are implemented this way. Scuba at Facebook, for example, is a system where data is intended to be sampled, so completeness of the data is not a requirement. Duplication, however, would not be acceptable. In this case, at most once is a more fitting processing guarantee than exactly once (Chen et al., 2016). Stylus is the only real-time system at Facebook designed with optimisations to provide at least once processing semantics. This is enabled by Swift's use of Scribe as a messaging system, which is backed by Swift for checkpointing. (Chen et al., 2016).

At Twitter, both Storm and its successor, Heron, offered at least once and at most once guarantees. Identified as a shortcoming by Kulkarni et al., (2015), the lack of exactly once semantics in Heron was recently addressed and implemented as "effectively once semantics". This means that data may be processed more than once (the topology would undergo a rewind in case of failure), but it is only delivered once (2018).

Netflix uses Kafka as its stream platform and messaging system (Wu et al., 2016), which means it provides inherent support for exactly once processing through idempotency and atomic transactions (Woodie, 2017).

3.5 Integration and Extensibility

This requirement refers to the capability to integrate with existing services and components. It also refers to provisions made to facilitate the extension of the existing architecture to incorporate different components in the future.

Although Facebook's real-time architecture is composed of many systems, they are integrated thanks to Scribe. Scribe works as a messaging system: all of Facebook's streaming systems write to Scribe, and they also read from Scribe. This allows for the creation of complex pipelines to cater for a multitude of use-cases (Chen et al., 2016). In terms of extensibility, any service developed to use Scribe as data source and output could integrate seamlessly with Facebook's architecture.

As part of a process to make Heron open-source, Twitter introduced a number of improvements to make it more flexible and adaptable to different infrastructures and use-cases. By adopting a general-purpose modular architecture, Heron achieved significant decoupling between its internal components, and increased its potential for adoption and extension by other companies (Fu et al., 2017).

Netflix's high level architecture is somewhat rigid in that there is no alternative to using Mesos as an orchestration and cluster management tool. Additionally, Titus must run as a single framework on top of Mesos. This limitation however was introduced by design. With Titus running as a single framework on Mesos, it can allocate tasks more efficiently, with visibility of resources across the entire cluster (Leung et al., 2017). At the time of writing, Titus is not yet open-source, so decoupling its components from Netflix-specific infrastructure and use-cases is not a requirement.

3.6 Distribution and Scalability

This requirement refers to the capability to distribute data processing amongst different machines, located in different data centres, in a multi-clustered architecture. Dynamic scaling, which addresses the possibility of adding or removing nodes to a running system without any downtime, is also addressed as part of this requirement.

Scalability was one of the driving factors behind the development of Scribe as a messaging system at Facebook. Similarly to Kafka, Scribe can be scaled up by increasing the number of buckets (brokers) running, thus increasing the level of parallelism (Chen et al., 2016). There is no mechanism in place for dynamic scaling of Puma and Stylus systems (Chen et al., 2016).

Heron was developed as a more efficient and scalable alternative to Storm. Heron, uses an in-house developed proprietary framework called Dhalion to help determine whether the cluster needs to be scaled up or down (Graham, 2017).

As Netflix's architecture is cloud-based, it is inherently elastic and scalable. Fenzo is responsible for dynamically scaling resources by adding or removing EC2 nodes to the Mesos infrastructure as needed (Schmaus et al., 2016).

3.7 Cloud Support and Elasticity

This requirement refers to the capability to move the architecture (or part of it) into the cloud to take advantage of the many benefits associated with its economies of scale. Elasticity in particular is a cloud property which allows a system to scale up and down according to demand. Since the user only pays for resources actually used, there is less wastage and it is theoretically cheaper than running the entire infrastructure locally with enough idle capacity to cover for any eventual spike. While scalability can be gained by increasing the number of nodes in any traditional architecture, it becomes much more powerful when combined with the elasticity of the cloud.

Based on the material examined, Neflix's architecture is the only which is predominantly cloud-based. Having started with services running on AWS virtual machines, they are now undergoing a shift towards a container-based approach, with a few services now running in containers on AWS infrastructure (Leung et al., 2017). Twitter has also undergone a shift towards a containerised architecture, albeit not cloud-based, with the development and implementation of Heron. As containers become more widespread, the risk of vendor lock-in is lowered, since containers enable the decoupling of the processing framework from the infrastructure they run in. Future migration to a safer multi-cloud setup is not only possible, but desirable (Vergilio and Ramachandran, 2018).

3.8 Fault Tolerance

This requirements refers to the capability of a system to continue to operate should one or more nodes fail. Ideally, the system should recover gracefully, with minimal repercussions on the user experience.

Fault-tolerance is a requirement of Facebook's real-time systems, currently implemented through node independence, and by using Scribe for all communication between systems. Scribe persists data to disk and is backed by Swift, a stream platform designed to provide checkpointing for Scribe. (Chen et al., 2016).

At Twitter, fault tolerance is addressed at different levels. At architectural level, a modular distributed architecture provides better fault tolerance than a monolithic design. At container level, resource provisioning and job scheduling are decoupled, with the scheduler responsible for monitoring the status of running containers and for trying to restart any failed ones, along with the processes they were running. At JVM level, Heron limits task processing to one per JVM. This way, should failure occur, it is much easier to isolate the failed task and the JVM where it was running (Fu et al., 2017). At topology level, the management of running topologies is decentralised, with one Topology Master per topology, which means failure of one topology does not affect others (Kulkarni et al., 2015).

As Netflix's production systems are cloud-based, fault tolerance is addressed from the perspective of a cloud consumer. The Active-Active project was launched by Netflix with the aim of achieving fault tolerance through isolation and redundancy by deploying services to the US across two AWS regions: US-East-1 and US-West-2 (Meshenberg et al., 2013). This project was later expanded to incorporate the EU-West-1 region, as European locations were still subjected to single points of failure (Stout, 2016). With this latest development, traffic could be routed between any of the three regions across the globe, increasing the resilience of Netflix's architecture.

3.9 Flow Control

This requirement refers to the capability to handle scenarios where the data source is emitting records faster than the system can consume. Architectures typically provide strategies for dealing with backpressure, e.g. dropping records, sampling, applying source backpressure, etc.

All real-time systems at Facebook read and write to Scribe. As described by Chen et al., this central use of a persistent messaging system makes Facebook's real-time architecture resilient to backpressure. Since nodes are independent, if one node slows down, the job is simply allocated to a different node, instead of the slowing down the whole pipeline (2016). The exact strategy used by Scribe to implement flow control is not made explicit in the paper.

Heron was designed with a flow control mechanism as an improvement over Storm, where producers dropped data if consumers were too busy to receive it. When Heron is in backpressure mode, the Stream Manager tightens the furthest upstream component (the spout) to slow down the flow of data through the topology. The data processing speed is thus reduced to the speed of the slowest component. Once backpressure is relieved and Heron exits backpressure mode, the spout is set back to emit records at its normal rate (Kulkarni et al., 2015).

Mantis jobs at Netflix are written using ReactiveX, a collection of powerful open-source reactive libraries for the JVM (Christiansen and Husain, 2013). RxJava, one of the libraries in ReactiveX originally developed by Netflix, offers a variety of strategies for dealing with backpressure such as, for example, the concept of a cold observable, which only starts emitting data if it is being observed, and at a rate controlled by the observer. For hot observables which emit data regardless of whether or not they are being observed, RxJava provides the options to buffer, sample, debounce or window the incoming data (Gross and Karnok, 2016).

3.10 Flexibility and Technology Agnosticism

This criterion refers to the capability of an architecture to use different technology in place of existing components.

Out of the three architectures investigated, Facebook's setup is the least flexible and the least technologically agnostic. With the exception of Hive and its ODS, built on HBase (Tang, 2012), Facebook's data systems were developed in-house to cater for very specific use-cases. This is perhaps the reason why, at the time of writing, only Scribe has been made open-source (Johnson, 2008), although it was not developed further, and the source-code is archived (Facebook Archive, 2014).

Heron's modular architecture is flexible by design, and the technologies chosen for Twitter's particular implementation, Aurora and Mesos are not compulsory for other implementations. Heron's flexibility is evidenced by its adoption by large scale companies such as Microsoft (Ramasamy, 2016), and its technology agnosticism is evidenced by its successful implementation on a Kubernetes cluster (Kellogg, 2017).

At programming level, Netflix is an active participant of the Reactive Streams initiative, which aims to standardise reactive libraries with an aim to rendering them interoperable. Considering that JDK 9, released in September 2017, is also compatible with Reactive Streams, there is potential for Mantis's jobs to be defined in standard Java.

At cloud infrastructure level, the use of containers as a deployment abstraction reduces the tight coupling between Netflix's artifacts and specific virtual machine offerings provided by AWS. This is defined by Leung et al., (2017) as a shift to a more application-centric deployment.

At architecture level, because Titus is not open-source, it is difficult to evaluate whether essential parts of its architecture such as the Mantis, Fenzo or the Mesos cluster could be replaced with an equivalent. Work however is under way to make the project open-source (Netflix TechBlog, 2017), which could attract important contributions from the community and enhance its flexibility and technology agnosticism.

4 CONCLUSION AND FUTURE WORK

This paper presented the results of a literature search for non-functional requirements relevant to real-world big-data implementations. Three companies were selected for this comparative study: Facebook, Twitter and Netflix. Their specific implementations of the non-functional requirements selected were compared and discussed in detail, and are summarised in this section.

Facebook and Twitter process the largest volume of data, with Twitter having the lowest requirement for latency. These two architectures were also explicitly designed to handle late and out of order data. In terms of processing guarantees, all three architectures support exactly-once semantics.

Although the existing systems at Facebook and Netflix are integrated, they were not designed as a unified modular framework. Heron, on the other hand, was developed by Twitter as an improvement over Storm, which suffered from bottlenecks and single points of failure. Heron's modular architecture makes it more flexible and technologically agnostic, as well as a stronger candidate for adoption by other companies.

Differently from Facebook and Twitter, which provide mechanisms for scalability and fault tolerance in their infrastructures, Netflix approaches this concept from a cloud consumer's perspective, since its architecture is cloud-based. Netflix's deployments are distributed over multiple regions, although support for multi-cloud is still lacking.

All three architectures provide mechanisms for flow control. Facebook and Twitter control backpressure from an infrastructure level, whereas Netflix provides methods and constructs to achieve this programmatically.

Our next step in this research is to use the non-functional requirements discussed in this study to guide the design and implementation of a new architecture for big data processing in the cloud: MC-BDP. MC-BDP is an evolution of the PaaS-BDP architectural pattern originally proposed by the authors. While PaaS-BDP introduced a framework-agnostic programming model and enabled different frameworks to share a pool of location and provider-independent resources (Vergilio and Ramachandran, 2018), MC-BDP expands this model by explicitly prescribing a pooled environment where nodes are deployed to multiple clouds.

ACKNOWLEDGEMENTS

This work made use of the Open Science Data Cloud (OSDC) which is an Open Commons Consortium (OCC)-sponsored project.

Cloud computing resources were provided by a Microsoft Azure for Research award.

Container and cloud native technologies were provided by Weaveworks.

REFERENCES

Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E. and Whittle, S. (2015) The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *Proceedings of the VLDB Endowment*, 8, pp. 1792–1803.

Akidau, T., Whittle, S., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D. and Nordstrom, P. (2013) MillWheel: Fault-Tolerant Stream Processing at Internet Scale. *Proceedings of the VLDB Endowment*, 6 (11) August, pp. 1033–1044.

Bernstein, P., Bykov, S., Geller, A., Kliot, G. and Thelin, J. (2014) *Orleans: Distributed Virtual Actors for Programmability and Scalability* [Online]. Available from: <https://www.microsoft.com/en-us/research/publication/orleans-distributed-virtual-actors-for-programmability-and-scalability/>.

Cao, L. (2017) Data Science: Challenges and Directions. *Communications of the ACM*, 60 (8) July, pp. 59–68.

Chen, G. J., Wiener, J. L., Iyer, S., Jaiswal, A., Lei, R., Simha, N., Wang, W., Wilfong, K., Williamson, T. and Yilmaz, S. (2016) Realtime Data Processing at Facebook. In: *Proceedings of the 2016 International Conference on Management of Data, 2016*. New York, NY, USA: ACM, pp. 1087–1098.

Cheng, B., Longo, S., Cirillo, F., Bauer, M. and Kovacs, E. (2015) Building a Big Data Platform for Smart Cities: Experience and Lessons from Santander. In: *2015 IEEE International Congress on Big Data, June 2015*. pp. 592–599.

Christiansen, B. and Husain, J. (2013) Reactive Programming in the Netflix API with RxJava. *Netflix TechBlog*, 4 December [Online blog]. Available from: <https://medium.com/netflix-techblog/reactive-programming-in-the-netflix-api-with-rxjava-7811c3a1496a> [Accessed 16 February 2018].

Chung, L. and Prado Leite, J. C. (2009) Conceptual Modeling: Foundations and Applications. Berlin, Heidelberg: Springer-Verlag, pp. 363–379.

Eliot, S. (2010) *Microsoft Cosmos: Petabytes Perfectly Processed Perfunctorily* [Online blog]. Available from: <https://blogs.msdn. microsoft.com/seliot/2010/11/05/microsoft-cosmos-petabytes-perfectly-processed-perfunctorily/> [Accessed 24 January 2018].

Facebook Archive (2014) *Scribe* [Online]. Facebook Archive. Available from: <https://github. com/facebookarchive/scribe> [Accessed 15 February 2018].

Fu, M., Agrawal, A., Floratou, A., Graham, B., Jorgensen, A., Li, M., Lu, N., Ramasamy, K., Rao, S. and Wang, C. (2017) Twitter Heron: Towards Extensible Streaming Engines. In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE), April 2017*. pp. 1165–1172.

Gianos, T. and Weeks, D. (2016) Petabytes Scale Analytics Infrastructure @Netflix [Online]. Presented at: *QCon, August 11, 2016, San Francisco*. Available from: <https://www.infoq.com/presentations/netflix-big-data-infrastructure> [Accessed 12 February 2018].

Graham, B. (2017) From Rivulets to Rivers: Elastic Stream Processing in Heron [Online]. Available from: <https://www.slideshare.net/billonahill/from-rivulets-to-rivers-elastic-stream-processing-in-heron> [Accessed 14 February 2018].

Gross, D. and Karnok, D. (2016) *Backpressure* [Online]. ReactiveX/RxJava Wiki. Available from: <https://github.com/ReactiveX/RxJava/wiki/Backpressure> [Accessed 15 February 2018].

Heron Documentation - Heron Delivery Semantics (2018) [Online]. Available from: <https://twitter.github.io/ heron/docs/concepts/delivery-semantics/> [Accessed 14 February 2018].

Johnson, R. (2008) Facebook's Scribe Technology Now Open Source. *Facebook Code*, 24 October [Online blog]. Available from: <https://code.facebook.com/ posts/214389698718537/facebook-s-scribe-technology -now-open-source/> [Accessed 15 February 2018].

Kellogg, C. (2017) The Heron Stream Processing Engine on Google Kubernetes Engine. *Streamlio*, 28 November [Online blog]. Available from: <https://streaml.io/blog/heron-on-gke-power-by-kubernetes/> [Accessed 15 February 2018].

Krishnan, S. (2016) Discovery and Consumption of Analytics Data at Twitter. 29 June [Online blog]. Available from: <https://blog.twitter. com/engineering/en_us/topics/insights/2016/discovery -and-consumption-of-analytics-data-at-twitter.html> [Accessed 9 February 2018].

Krishnan, S. and Tse, E. (2013) Hadoop Platform as a Service in the Cloud. *The Netflix Tech Blog*, 10 January [Online blog]. Available from: <http://techblog.netflix. com/2013/01/hadoop-platform-as-service-in-cloud.ht ml> [Accessed 30 October 2016].

Kulkarni, S., Bhagat, N., Fu, M., Kedigehalli, V., Kellogg, C., Mittal, S., Patel, J. M., Ramasamy, K. and Taneja, S. (2015) Twitter Heron: Stream Processing at Scale. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015*. New York, NY, USA: ACM, pp. 239–250.

Leung, A., Spyker, A. and Bozarth, T. (2017) Titus: Introducing Containers to the Netflix Cloud. *Queue*, 15 (5) October, pp. 30:53–30:77.

Meshenberg, R., Gopalani, N. and Kosewski, L. (2013) Active-Active for Multi-Regional Resiliency. *Netflix TechBlog*, 2 December [Online blog]. Available from: <https://medium.com/netflix-techblog/active-active-for-multi-regional-resiliency-c47719f6685b> [Accessed 15 February 2018].

Pearce, J. (2013) *2013: A Year of Open Source at Facebook* [Online]. Facebook Code. Available from: <https://code.facebook.com /posts/604847252884576/2013-a-year-of-open-source-at-facebook/> [Accessed 12 February 2018].

Peng, B. (2017) *[ISSUE-1124] - Windows Bolt Support #2241* [Online] [Heron]. Twitter, Inc. Available from: <https://github.com/twitter/heron/pull/2241> [Accessed 12 February 2018].

Ramasamy, K. (2016) Open Sourcing Twitter Heron. *Twitter Engineering Blog*, 25 May [Online blog]. Available from: <https://blog.twitter.com/engineering/ en_us/topics/open-source/2016/open-sourcing-twitter-heron.html> [Accessed 15 February 2018].

Schmaus, B., Carey, C., Joshi, N., Mahilani, N. and Podila, S. (2016) Stream-Processing with Mantis. *Netflix TechBlog*, 14 March [Online blog]. Available from: <https://medium.com/netflix-techblog/stream-processing-with-mantis-78af913f51a6> [Accessed 31 January 2018].

Stout, P. (2016) Global Cloud—Active-Active and Beyond. *Netflix TechBlog*, 30 March [Online blog]. Available from: <https://medium.com/netflix-techblog/global-cloud-active-active-and-beyond-a0fdfa2c3a45> [Accessed 15 February 2018].

Tang, L. (2012) Facebook's Large Scale Monitoring System Built on HBase [Online]. Presented at: *Strata Conference + Hadoop World, October 24, 2012, New York, NY, USA*.

Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S. and Ryaboy, D. (2014) Storm@Twitter. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, 2014*. New York, NY, USA: ACM, pp. 147–156.

Tse, E., Luo, Z. and Yigitbasi, N. (2014) Using Presto in Our Big Data Platform on AWS. *The Netflix Tech Blog*, 10 July [Online blog]. Available from: <https://medium.com/netflix-techblog/using-presto-in-our-big-data-platform-on-aws-938035909fd4> [Accessed 12 February 2018].

Twitter, Inc. (2018) *Scalding: A Scala API for Cascading* [Online]. Twitter, Inc. Available from: <https://github.com/twitter/scalding> [Accessed 12 February 2018].

Updates on Netflix's Container Management Platform (2017) *Netflix TechBlog*, 14 November [Online blog].

Available from: <https://medium.com/netflix-techblog/updates-on-netflixs-container-management-platform-a91738360bd8> [Accessed 16 February 2018].

Vergilio, T. and Ramachandran, M. (2018) PaaS-BDP - A Multi-Cloud Architectural Pattern for Big Data Processing on a Platform-as-a-Service Model. In: *Proceedings of the 3nd International Conference on Complexity, Future Information Systems and Risk* - Volume 1: COMPLEXIS, ISBN 978-989-758-297-4, pp. 45-52.

Woodie, A. (2017) A Peek Inside Kafka's New 'Exactly Once' Feature. *Datanami*, 7 March [Online blog]. Available from: <https://www.datanami.com/2017/07/03/peek-inside-kafkas-new-exactly-feature/> [Accessed 14 February 2018].

Wu, S., Wang, A., Daxini, M., Alexar, M., Xu, Z., Patel, J., Guraja, N., Bond, J., Zimmer, M. and Bakas, P. (2016) *The Netflix Tech Blog: Evolution of the Netflix Data Pipeline* [Online]. Available from: <http://techblog.netflix.com/2016/02/evolution-of-netflix-data-pipeline.html> [Accessed 30 October 2016].