Reactive Architectures

A model for modern software needs

Damien Murphy
Higher Diploma in Science, Computer Science
Waterford Institute of Technology
Waterford, Ireland
damienomurchu@gmail.com

Eamonn Kenny
Higher Diploma in Science, Computer Science
Waterford Institute of Technology
Waterford, Ireland
amen_cionna@hotmail.com

Abstract— **Significant changes in the technological landscape in recent years have resulted in new challenges and considerations for those involved in the design of modern software. Advancements in hardware, internet growth, and the IoT age have resulted in unparalleled levels of usage. To answer the critical challenges of extreme usage whilst meeting the expectation levels of modern users, traditional approaches have been severely tested and frequently found lacking.**

For many companies this increase in traffic coupled with the ambition to deliver a consistent user experience, under any circumstances, has called for the implementation of a reactive architecture, which has demonstrated the ability to deliver the flexibility, scalability and vastly improved fault tolerance required by large scale organizations.

Responsive, scalable, resilient applications are the aim of reactive architectures as the importance of software in everyday life increases. Scaling up can be costly and inefficient endeavour, and scaling out may not be straightforward. In contrast, reactive individual services or node clusters of services are scaled out as required, and back as demand cools down. Small, independent services, internally able to handle faults provide a quick and positive user experience.

The choice of architecture for an application, however, is not always clear cut as companies must weigh their needs against the available designs. This paper will explore the nature of reactive designs and what they may offer. (*Abstract*)

Keywords—reactive; architectures; systems; responsive; micro-services;

I. Introduction

"Today applications are deployed on everything from mobile devices to cloud-based clusters running thousands of multi-core processors. Users expect millisecond response times and 100% uptime. Data is measured in Petabytes. Today's demands are simply not met by yesterday's software architectures." [1]

Reactive architectures have come to the fore as a response to the unrelenting growth in the field of technology over the last decade. Selecting the correct architecture to service the needs of a business is more important now than the selection of any framework and while new approaches can be the solution for some requirements we will examine where the reactive solution is most appropriately adopted. This paper will explore reactive systems in terms of their defining principles: responsiveness, scalability, resilience and message-driven design, providing examples of where the design has proved of most benefit.

"Monolithic applications are relatively quick to stand up, easier to deploy and are well known entities that most tools, frameworks and IDEs are designed to deal with. The weaknesses of a monolithic application tend to show as it matures." [2]. Despite care taken in their design they hold the potential to stifle development as they grow, with staff coming and going as the application ages, each adding new interdependencies through their work, taking with them detailed knowledge of the workings of the system when they move on. This can lead to complications when adding new features as one change anywhere requires a full rebuild and is costly in time taken to deploy.

With the reactive ideology of breaking applications into smaller, self-contained components, each being a feature or function of the business, deployment becomes easier as only a specific service is deployed with the services themselves scaled individually as required by demand. This segregation makes understanding the functionality of each service easier while also allowing for new developers to become familiar with any particular service relatively quickly.

"Like many technology decisions, the choice is not clear-cut. The architecture appropriate for an organization comes down to assessing the application, the team and even the organization itself. This generally encourages a monolith-first approach given the reduced application overhead for documentation, testing, infrastructure and orchestration." [3]. For large scale companies, receiving vast amounts of traffic, reactive architecture provides a suitable development path for the future while smaller scale companies, without the benefit of a sizeable developer operations team to manage the larger ecosystem which occurs as a by-product of the reactive approach, may appropriately elect a monolithic design.

II. The Need for Reactive

The case for reactive systems stems primarily from a changed landscape. Sustained and explosive growth in user numbers, volumes of data and the increasing proliferation of network connected devices have led to unexpected levels of user expectation and pressure on modern software systems [4].

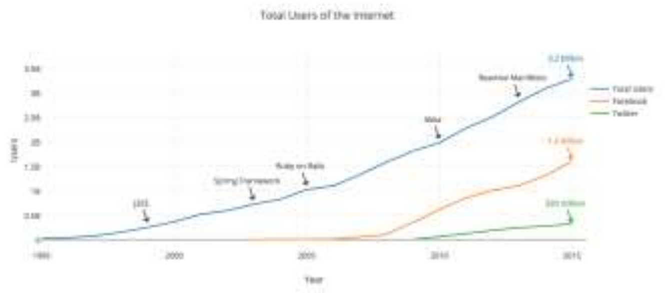

Fig. 1. Number of internet users, 1995- 2015

In coping with this often impossible cocktail of demands, traditional software approaches and tooling has often been found wanting. To fill the void, what have been collectively referred to as reactive approaches have emerged. This umbrella of technologies and approaches represents a new paradigm for software development.

What reactive applications offer is a proven ability to handle large volumes of activity and data streams, as well as promises of software that is both easier to develop and more amenable to change. With paradigms of the past not always scaling to demands of the present, reactive architectures are regarded to be an important element in the future of technology.

III. Impact of Reactive Approaches

Adoption of reactive approaches has led to several key impacts in modern software. The area of UI has been the most noticeable, with the production of compelling applications, simplification of UI code and approaches, and more broadly the elevation of the user flow to the status of a first class citizen as opposed to something scripted by the software developer. The impact has been particularly strongest where the UI responds to flows and changes in streams of data.

The front end has not been the only area impacted, ripple effects have also been felt on the server side, seeing servers evolve from simple providers of static web pages to the rendering of completely dynamic, server-side generated pages.

Successful adoptions of reactive approaches can be seen in the case of Twitter and Walmart Canada.

A. Twitter

During the 2010 World Cup Twitter hosted a real time global conversation that brought it to the brink of its resources. “The influx of Tweets, from every shot on goal, penalty kick, yellow or red card, repeatedly took its toll and made Twitter unavailable for short periods of time”. [5] At the time, peak activity was recorded as 3,283 tweets per second. Continued growth hampered any efforts by Twitter to resolve these capacity issues, which were finally handled when they moved to a reactive structure. An increase in capacity from the ability to handle 2-300 requests per second per host to 10-20,000 illustrates the efficiencies a reactive approach can deliver.

B. Walmart Canada

“We saw immediate improvements as the Scala stack came online that translated into faster response times and higher conversion rate.” [6] Simon Rodrigues VP of ecommerce at Walmart Canada.

Driven by an intention of improving the online shopping experience of their customers across a variety of hardware devices; phones, tablets, laptops etc., Walmart moved to a reactive design which saw not only an improvement in service, but delivered reductions in backend costs while maintaining resilience and scalability. In particular, greater capacity and speed in serving pages, faster development, and the ability to distribute its systems over a cloud-based infrastructure were pointed to as key benefits of its change to a reactive architecture.

IV. Reactive Manifesto

Published in September 2014, The Reactive Manifesto was a document produced by many in the industry which has attempted to define a common vocabulary with which to discuss reactive applications. A dictionary of best practices, the manifesto provided a set of key principles and their building blocks to guide the design of reactive applications [7].

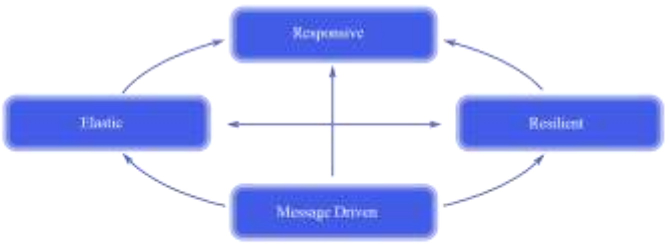

Fig. 2. Elements of a reactive application

The Manifesto asserts that with responsive applications being the aim, this is achieved by ensuring the software is both resilient and scalable, both of which depend on an underlying message-driven foundation [8].

A. Responsive

Responsiveness is the end goal of any reactive application, and a system should remain responsive and continue to deliver a consistent user experience, irrespective of unexpected events such as infrastructure failures or large traffic spikes. To achieve this goal a reactive system needs to possess both resilience and scalability. A message-driven architecture provides the overall foundation for such a responsive system.

B. *Resilience*

In order to be considered resilient, a modern application needs to be able to handle failures. Resilience becomes a trickier problem however when considering the nature of today's applications, which are commonly composed of other applications and are integrated via web services and network protocols. Resilience ensures an application can survive failures to these integral components, and still continue to offer the user a consistent and positive experience.

Fig. 3. Various factors for a resilient application

Effective design and architecture principles are essential elements in ensuring such resiliency, and while resiliency has often been a weak link of many systems, it can no longer afford to be considered an afterthought if a system is to remain responsive in less than ideal conditions.

Resiliency has many facets, performance, endurance, and security key amongst them. All these facets need to be considered when contingency planning, not just some.

C. *Scalability*

Going hand-in-hand with resilience, scalability ensures a system is easily upgraded on demand so it remains responsive regardless of load conditions. Scaling up and scaling out offer the options of how to expand resources, with scaling out on demand being the ultimate goal of a reactive application.

Scaling is of critical importance with traffic spikes usually representing positive consequences for a site, be it increased e-commerce traffic, or spikes in user growth and adoption. Being able to handle these spikes while offering a consistently positive user experience is where scalability comes in.

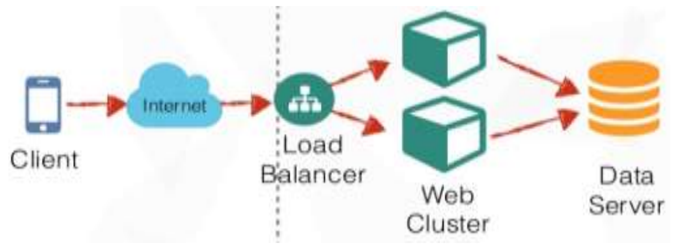

Fig. 4. A typical scaling model

Thread-based concurrency presents a number of critical and tricky scaling-based issues, and while asynchronous concurrency is not a requirement of reactive systems, it is often adopted to avoid such scaling issues.

D. *Message-Driven*

A message-driven architecture is the foundation of reactive applications, and may be event-driven, actor-based, or a combination. Both methods eschew the use of a traditional call stack in favour of lightweight asynchronous message passing, and differ at their most basic with events potentially being observed by multiple observers, while with actors, events are always directed to one specific destination.

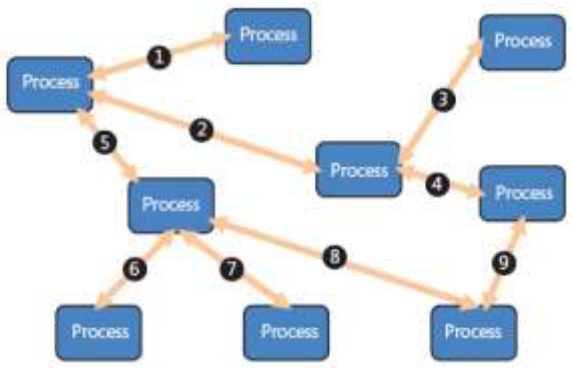

Fig. 5. Example of a message-driven system

The main consequence of an event-driven architecture is the phenomenon of callback-hell, with recipients of messages being anonymous callbacks instead of addressable recipients. Actor-based concurrency in comparison avoids callback-hell by explicitly directing messages to specific actors. An actor-based architecture may also make scaling out computation across network boundaries easier, offering loose coupling of components, with the calling code only needing to be called back if necessary, thus opening up many possibilities by not tying applications to a single space in memory.

V. TYPES OF REACTIVE APPLICATIONS

Given the broad possible scope of types of reactive applications, several key dimensions can be used to distinguish the members of the reactive application family; synchrony, determinism, and update process [9].

A. *Synchrony*

The model of synchrony and how a reactive design chooses to handle synchronization of data flowing across the application can often distinguish reactive applications. The adoption of asynchronous communications results in greater responsiveness, and also increases in the efficiency of servers, as server time can be utilized much more productively.

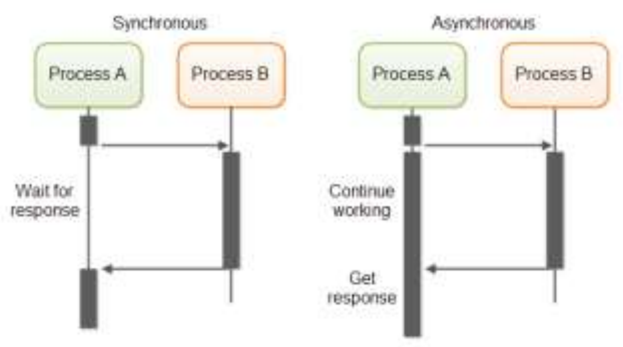

Fig. 6. A synchronous vs asynchronous system

B. Determinism

At its basic level, determinism is whether a function returns the same values when fed the same inputs, assuming the database also remains in the same state during both function calls. This choice may affect the language chosen for a reactive application.

C. Update Process

Does the reactive application employ callbacks, dataflow, or actors to manage the processing and updating of data across the application? Choice of update process may not be trivial and refinement in these areas, especially of callbacks, is ongoing.

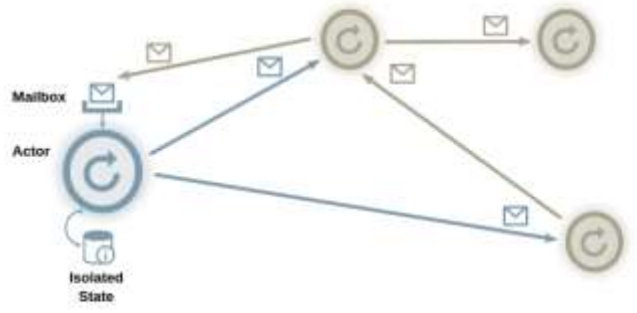

Fig. 7. Example of an actor-based system

VI. Caveats of Reactive Adoption

As a new technology, there has been a lot of excitement around reactive approaches. As with any software problem, no one approach is suitable for all applications. Still being fully understood, there is a risk of a reactive approach being adopted where it may be unsuitable, or simply reactive approaches being over-used or misapplied.

Over and mis-adoption of reactive approaches may lead to the introduction of unnecessary complexity into an application, primarily stemming from the area of asynchronous message-passing. The suggested solution is to avoid implementing reactive solutions in applications where they are unsuited, where an alternative approach may be more suitable, or simply where the adoption of a reactive approach does not make sense in terms of the benefits it yields.

VII. Conclusions

Scenarios such as Black Friday are no longer bringing websites to the brink of their resources. Reactive approaches enable extra services that are easily created and removed as required to deal with spikes in consumer traffic. With the exception of malicious acts, large traffic spikes generally indicate a company is doing something very positive, making the ability to remain responsive at all times of critical importance. Reactive systems allow a system to do just that, even in the face of unexpectedly large traffic spikes.

Examples of reactive success stories trickle out with regularity, and companies like Twitter and Walmart Canada cited earlier are just two of many. Moving to a reactive architecture has provided a promising way forward for ecommerce, social media and social networking companies amongst many others. The ability to scale on demand provides a powerful benefit for any company in handling growth, traffic spikes, or managing their backend infrastructure costs efficiently.

In conclusion, reactive approaches occupy an important role in the future of software design, and aside from other benefits they may bring, bear particular consideration for any system where data flows and traffic are a key element.

Footnotes

[1] Various, The Reactive Manifesto, Sept. 2014 [online]. Available: http://www.reactivemanifesto.org/ [Accessed: 27-Nov-2016]

[2] Thomas Overton, Microservices Versus Monoliths Nov. 2015 [online] Available: https://www.sumologic.com/blog-devops/microservices [Accessed: 27-Nov-2016]

[3] Thomas Overton, Microservices Versus Monoliths Nov. 2015 [online] Available: https://www.sumologic.com/blog-devops/microservices [Accessed: 27-Nov-2016]

[4] K. Webber, What is Reactive Programming?, Aug. 2014 [online]. Available: https://medium.com/reactive-programming/what-is-reactive-programming-bc9fa7f4a7fc#.yj80539us [Accessed: 27-Nov-2016]

[5] Raffi Krikorian, New Tweets per second, and how! Aug 2013 [online].Available: https://blog.twitter.com/2013/new-tweets-per-second-record-and-how [Accessed 27-Nov-2016]

[6] Unknown, Walmart Boosts Conversions By 20% With Lightbend ReactivePlatform[online].Accessible:https://www.lightbend.com/resources/case-studies-and-stories/walmart-boosts-conversions-by-20-with-lightbend-reactive-platform [Accessed 27-Nov-2016]

[7] R. Kuhn, What is the significance of the manifesto? May 2015 [online]. Available: https://www.quora.com/What-is-the-significance-of-the-Reactive-Manifesto [Accessed: 27-Nov-2016]

[8] Various, The Reactive Manifesto, Sept. 2014 [online]. Available: http://www.reactivemanifesto.org/ [Accessed: 27-Nov-2016]

[9] Various, Reactive Programming, Nov. 2016 [online]. Available: https://en.wikipedia.org/wiki/Reactive_programming [Accessed: 27-Nov-2016]

REFERENCES

Various, Reactive Programming, Nov. 2016 [online]. Available: https://en.wikipedia.org/wiki/Reactive_programming [Accessed: 27-Nov- 2016]

Various, The Reactive Manifesto, Sept. 2014 [online]. Available: http://www.reactivemanifesto.org/ [Accessed: 27- Nov- 2016]

R. Kuhn, What is the significance of the manifesto? May 2015 [online]. Available: https://www.quora.com/What-is-the-significance-of-the-Reactive-Manifesto [Accessed: 27- Nov- 2016]

Unknown, Walmart Boosts Conversions By 20% With Lightbend ReactivePlatform[online].Accessible:https://www.lightbend.com/resources/case-studies-and-stories/walmart-boosts-conversions-by-20-with-lightbend-reactive-platform [Accessed 27-Nov-2016]

Raffi Krikorian, New Tweets per second, and how! Aug 2013 [online].Available: https://blog.twitter.com/2013/new-tweets-per-second-record-and-how [Accessed 27-Nov-2016]

K. Webber, What is Reactive Programming?, Aug. 2014 [online]. Available: https://medium.com/reactive-programming/what-is-reactive-programming-bc9fa7f4a7fc#.yj80539us [Accessed: 27-Nov- 2016]

Thomas Overton, Microservices Versus Monoliths Nov. 2015 [online] Available: https://www.sumologic.com/blog-devops/microservices [Accessed: 27-Nov-2016]

Steve Pember, Lecture, Why Reactive architecture will take over the world. April 2014 [online]
Available: https://www.youtube.com/watch?v=0oovNxZnkAE [Accessed: 1-Dec-2016]

Matt Graves, The 2010 World Cup: A Global Conversation. July 2010 [online] Available: https://blog.twitter.com/2010/the-2010-world-cup-a-global-conversation [Accessed: 1-Dec-2016]