Declarative Distributed Stream Processing

PhD Stage 2 Report

Jonathan Dowland <jon.dowland@ncl.ac.uk>, June 2021

I have been studying **part-time** towards a PhD since 2017/18. This report marks the end of Stage 2 (my 4th year of study). In this report I outline the work completed during this stage, the work remaining and my plan for continuing to completion.

1 Background

1.1 Distributed Stream-Processing

Many modern applications, in domains ranging from smart cities to healthcare, have a requirement for the timely processing of data arriving at high speed, for example data generated by sensors in the Internet of Things (IoT). Such systems may need to meet a range of other non-functional requirements, including: reliability; security; energy efficiency (for example to prolong the battery life of sensors in the field), or privacy (to remove or de-personalise data prior to transmitting it over open networks).

The combination of requirements, very high data arrival rates, and the desire for timely processing, makes the design and management of the supporting infrastructure very challenging. The current generation of IoT tools adopt the principles of stream processing and are designed around a three-tier architecture: sensors generate data, that is sent on to a local gateway (e.g. a smartphone or field gateway) before being passed on to the Cloud for processing.

However, in some domains it can be beneficial to perform some processing on the gateway or on the sensors themselves[MW17], so as to reduce the volume of data sent onwards to the cloud; or to reduce the frequency with which sensors must invoke their networking hardware, thus reducing energy expenditure; or to avoid transmitting sensitive data across public networks, by filtering or anonymising data sets at the point of collection.

Modern distributed stream-processing systems attempt to separate the functional definition (typically specified as a software program) and the non-functional requirements (the deployment environment and constraints such as power requirements, network utilisation limits, etc.), with varying degrees of success. Typically the programmer must consider and address both their functional and non-functional requirements when authoring their stream-processing program.

We are exploring a different approach where the stream-processing operations and the non-functional requirements are described separately as inputs to an Optimiser, which automatically generates the most appropriate deployment for the available resources, which may include sensors and gateways, meeting both the functional and non-functional requirements.

An initial approach (by PhD student Peter Michalák[MW17]) used an extended version of SQL as the method of describing the computation. However, SQL has limited expressivity which proved a barrier to encoding some stream-processing operations.

In contrast, in our project we are exploring using functional programming to describe the computation.

1.2 Purely-Functional Programming

Functional Programming (FP) is a software development paradigm where the principle building blocks of programs are functions[Bir14] (as opposed to e.g. abstract objects in Object-Oriented Programming) and programs are composed declaratively using expressions, rather than imperatively with sequences of statements.

Advocates of FP believe that many of its properties have advantages for the design and implementation of large and complex software systems[Cup89]. Constructing systems declaratively results in the programmer focussing on *what* a system should do, rather than the minutiae of *how* the work should be performed.

Purely-functional programming is a variant of FP where the behaviour of functions is entirely defined in terms of their input arguments and output value, and they can perform no other actions (referred to as *side-effects*). Consequently, purely-functional expressions are *referentially transparent*, and can be substituted for any other expression which evaluates to the same value for the same inputs.

Referential transparency enables *equational reasoning*, a technique for transforming functions through a process of substitution by applying laws or rules[Bir14].

1.3 Purely-Functional Stream Processing

My research aim is to establish to what extent the advantages of purely-functional programming can be applied to the design and operation of a distributed stream-processing system.

The declarative nature of purely-functional programming allows for a high degree of abstraction. I am investigating whether this enables the construction of a system where the user can specify the functional behaviour of the program completely independently from the deployment and non-functional requirements.

In order to meet the non-functional requirements, it may be necessary to adjust the stream-processing program whilst preserving the functional behaviour. Equational reasoning is a powerful tool for encoding program transformations and can be used to build rewriting systems[PTH01]. I am investigating whether rewrite rules are expressive enough to enable the design of an Optimiser that encodes and applies useful transformations for stream-processing optimisation.

A distributed stream-processing system needs to partition and distribute a stream-processing definition onto individual nodes. I am investigating whether any facets of purely-functional programming are particularly beneficial for the design and implementation of a Partitioner, and how automated partitioning should interact with the Optimiser.

1.4 Foundations

We have developed a prototype named *StrIoT*[aut20] in the *Haskell* purely-functional programming language. Stream-processing in *StrIoT* is defined in terms of a restricted set of functional operators with well-understood semantics. This prototype has three distinct components: a Partitioner to divide a stream-processing program into disjoint sub-programs; Library code to support execution and inter-connection of the sub-programs across multiple compute nodes; and an Optimiser.

My research is focussed on the optimiser, partitioner and deployer (another PhD student — Adam Cattermole — is also working on the stream processing library, but his focus is different: it is on run-time adaptivity).

2 Work completed

Some of the work described here is covered in more depth in my Stage 2/Year 3 interim report[Dow20].

2.1 Engineering

I spent the majority of my time over the last Stage completing the engineering work for *StrIoT*. I designed and implemented the *Logical Optimiser* and *Automatic Partitioning* components and integrated them with the *Deployer* to create an end-to-end system. This system applies a cost model (described below) to determine the best program variant and partitioning scheme.

A high-level overview of the *StrIoT* architecture is provided in Figure 1. The grey boxes describe components that are not implemented.

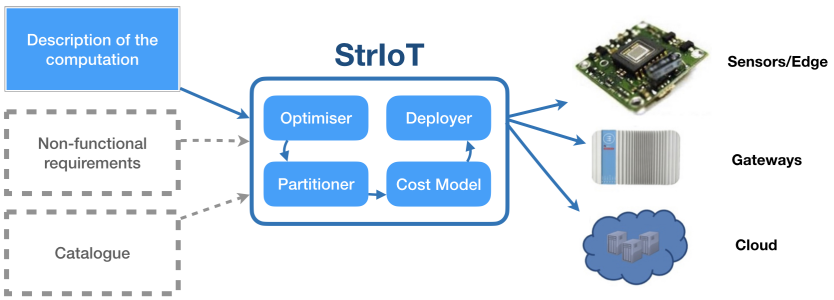

Figure 1: StrIoT architecture diagram

2.2 DEBS Paper

Paul Watson, Adam Cattermole and I wrote and submitted a paper to the 14th ACM International Conference on Distributed and Event-based Systems [Var20].

The paper is an overview of the *StrIoT* system and work completed to-date. My main contribution is the section on logical optimisation and deployment.

During the process of writing the paper we decided that we would be unable to complete the work needed to implement a Cost Model into our proof-of-concept system in the time available to us before the submission deadline. For this reason we instead focussed on ensuring the other components were completed and functioning to a high standard.

The paper was rejected but we received very constructive feedback from the reviewers. The lack of a cost model was highlighted. I describe the remaining feedback in Section 3.1.

2.3 Cost Model

Subsequent to submitting the DEBS paper, I developed an initial cost model that models stream-processing programs using queueing theory and calculates the cost of a pairing of a stream-processing program (derived by the *Logical Optimiser*) and a partitioning

(generated by the *Automatic Partitioner*). The utilisation of each logical Operator is calculated and the cost is based on the sum total of utilisations assigned to each partition.

I have now completed the design and implementation of the system I need in order to explore my research question.

3 Work remaining

3.1 Revised DEBS paper

The paper we submitted to DEBS 2020 was rejected but we received some very helpful feedback from the reviewers. Now that I have a running system with a basic cost model, automatic partitioning, automatic partition deployment and an optimiser, I will re-work the paper, address the concerns raised by the reviewers and submit to a future conference in the next 6 months. Addressing the reviewer's concerns will improve the quality of both my research and my thesis.

3.1.1 Consistent example problem

We were not able to use a single example problem throughout the paper. We opened with simple examples in the IoT domain, but switched to alternative problems for later sections. This was identified as a source of confusion by reviewers.

Our *StrIoT* implementation also lacks an encoding of a stream-processing program for a real-world problem which is demonstrably improved by the rewriting or partitioning processes.

I will therefore devise an example stream-processing program within a real-world problem domain, such that processing it with *StrIoT* results in a clearly improved program, in terms of a non-functional requirement of relevance to the domain. I will then use this example program consistently throughout the revised paper.

3.1.2 Design justification

We did not sufficiently explain the rationale behind some of our design choices, such as the use of purely-functional programming, Haskell as the implementation language, or the choice of the restricted operators from which users can compose their programs. Some reviewers were unsure what was novel about our approach, pointing out that the choice of operators are common to many existing systems.

These issues can be addressed by revising the existing material to make justifications more clear. In particular, our Logical Optimiser is only possible due to the choice of a purely-functional language.

I will re-write the *Related Work* section of the paper to include comparisons to existing stream-processing systems in use in the field. In order to support this work, I will first revisit surveying all such systems and their relevant properties.

3.2 Further cost models

My initial Cost Model is intended to quickly demonstrate the efficacy of *StrIoT* for the revised DEBS paper and is relatively modest. It's possible that a different example problem (described above) will not be enough to show strong results, and the existing cost model may need further refinement.

Additionally, alternative models could be developed to reason about other non-functional requirements, such as constraints on bandwidth between physical nodes. We already

model event arrival rates. This could be achieved by also considering the sizes of event data.

The current cost model relies upon a queueing theory model which has a set of assumptions that must hold for the calculations to be valid. These include constraints on the distribution of arrival rates of events throughout the network. Some of the stream-processing operators, such as filters and windowing, can break these assumptions. Further work on cost models could look to address these issues. Avenues to explore include chains of queueing networks and Kingsman correction. We are in discussion with Paul Ezhilchelvan on this topic.

3.3 Paper on derivation of rewrite rules

I plan to write a separate paper based on the work described in [Dow19a], Section "Derivation of stream rewriting rules" and encoding and categorisation described in [Dow20], Section 2.1.

Throughout the last Stage I prioritised preparing the DEBS paper and the elements of my work that were required to support it. Work on a rewrites paper was on hiatus. I did return to my notes on rewrite rules from time to time as and when my main worked touched on that area or when I had a realisation about a rewrite rule or their properties.

The most significant new work on rewrite rules was beginning to map each rule to a category of known stream-processing optimisations[Hir+14]; describe where some rewrite rules cause re-ordering of the input events and recognising that the flexibility afforded to user-supplied windowing functions made analysis of windowing difficult.

One area to expand on is to begin analysis of specific window-maker functions supplied with *StrIoT* (sliding or non-overlapping windows based on event times or number of events) as we can reason further about their behaviour than other unknown, user-supplied functions.

3.3.1 QuickSpec

QuickSpec[Sma+17] is a system for discovering rules and laws from a set of pure functions. If time allows in the PhD, It would be interesting to see whether QuickSpec or a similar tool could be used to derive further logical rewrite rules that I did not discover in my systematic, manual operator comparison[Dow19a].

3.4 Catalogue

StrIoT currently assumes the deployment targets are homogeneous. In order to optimise for deployments in an environment where different types of physical node have different properties, for example low-powered sensors which may not be able to perform complex computation such as described in Section 1.1, the *Catalogue* will need to be designed and implemented.

The Catalogue could also describe properties of the links between nodes, allowing us to model links that are bandwidth-limited (perhaps for cost reasons) and extent *StrIoT* to optimise for bandwidth constraints.

4 Plan

My main focus in the last Stage was the preparation of our Paper for DEBS 2020 ("Paper #1" in my original plan[Dow19a]). I found it especially effective (and motivating) to

organise my work towards that milestone. I'm therefore structuring my remaining work around similar milestones which are described below.

The broad outline of my plan is summarized in the GANTT chart in Figure 2.

4.1 Plan review

When I wrote my Plan in June 2019 I tried to consider all possible risks to the project, but I failed to consider an international Pandemic, which has had a significant impact on my work. For health reasons, following Government advice, I spent most of 2020 isolating together with my immediate family. I am very fortunate that I was able to continue PhD work during this time, although it was (and remains) very difficult to predict how much time I will be able to assign to PhD studies on a week-by-week basis.

By this point in time, my original plan forecast that Implementation and Testing work would be complete, and two papers written.

Implementation is largely complete, although I expect a continued stream of low-level maintenance work and bug fixing to continue throughout the remaining time. An initial Cost Model is in place. The bulk of any further implementation work will be on encoding improved cost models.

I wrote and submitted one of the two forecast papers, and opted to focus on that work and pause work on the second paper.

4.2 Milestones

4.2.1 Revised DEBS paper

This milestone collects the majority of the work required for the core of my PhD (described in Section 3.1). An important initial task is to select the conferences or journals to target and determine the relevant deadlines. Assuming we target DEBS 2022: Submission deadline is usually February/March for the conference taking place the following June/July. Working on this basis I have up to approximately 6 months (Jul 2021 to Feb 2022) to complete the supporting work and write the paper.

4.2.2 Rewrite rules paper

As discussed in Section 3.3, I plan to write up my work on program rewriting rules. This is a lower priority than the revised DEBS paper as the relevant work that contributes to the core of my PhD is already complete. I have enough material to form the basis of this paper, but depending on the rate of progress of my core objectives, and whether I choose to pursue an extension activity in the area of rewrite rules, the scope of this paper could be expanded. I shall first identify the conferences or journals I plan to submit to, in order to establish a submission deadline to work to.

4.2.3 Further cost models

As described in Section 3.2, there are a number of avenues for improving the cost model. Which route to take depends upon how well work goes on the initial cost model and queueing theory and whether our simple cost model to be described in the revised DEBS paper is sufficient to publish some results.

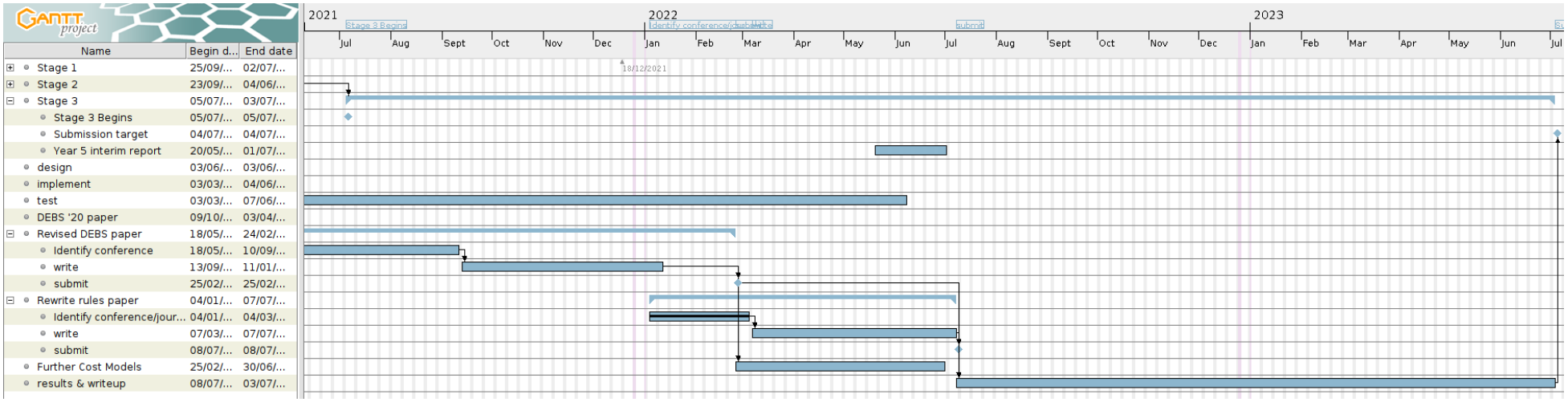

Figure 2: Project GANTT Chart

5 Thesis Outline

Early on in my research I focussed on the logical optimiser aspect of the architecture and attempting to prove its value has become the focus of my PhD.

The rewrite rules, their derivation, analysis and applying them to stream-processing programs is one main piece of work that is complemented by demonstrating their effectiveness with a robust cost model grounded in a sound theory.

My draft thesis outline below reflects the importance of those two main aspects of my work.

5.1 Draft outline

1. Abstract
2. List of Research Outputs
3. Introduction
4. Related Work
    4.1. Stream-processing systems
    4.2. Purely-Functional programming
    4.3. Queueing theory
5. StrIoT architecture overview
6. The Optimiser and Partitioner
7. Cost Models
8. Evaluation
9. Conclusions
    9.1. Thesis summary
    9.2. Contributions
    9.3. Future research directions
10. Appendices

6 Research outputs

6.1 *StrIoT* open source software

The primary output from my work so far is the research stream-processing platform *StrIoT*, which is open source software. I have made many contributions to *StrIoT* within this stage, which have been organised into 40 separate GitHub "Pull Requests"[Dow21b].

6.2 DEBS 2020 paper submission

As described in Section 2.2, I co-wrote a paper which we submitted to 14th ACM International Conference on Distributed and Event-based Systems [Var20] which was unfortunately rejected.

The paper is available upon request.

6.3 Personal blog posts

In order to practice technical writing, explore how to summarize and present the area of focus at a given time and to get early feedback from relevant online communities, I have been blogging about my PhD work. In this Stage I have written eleven blog posts[Dow21a]. These have attracted some useful comments and suggestions in particular from the Haskell functional-programming community. On three occasions my blog posts were picked up and re-broadcast by "Haskell Weekly News", a popular community newsletter[Faua][Faub][Fauc].

6.4 PhD Poster Session

I prepared and submitted a poster to the School of Computing PhD candidates poster session in 2019[Dow19b]. I adopted the "Better Poster" format[Mor19], substituting the "main finding" as the aspect given the highest priority for my research question, which is more fitting for the stage of my research.

6.5 Short presentation for Dr. Paul Ezhilchelvan

I began working with Dr. Paul Ezhilchelvan as I started exploring Queueing Theory as the theoretical underpinnings for a cost model. I prepared a short, specific presentation[Dow21c] for Dr. Ezhilchelvan which aimed to provide a brief introduction to the design of *StrIoT* and how I was attempting to its concepts into queueing theory.

References

[aut20] *StrIoT* authors. *StrIoT*. 2020. URL: https://github.com/striot/striot.

[Bir14] R. Bird. *Thinking Functionally with Haskell*. Cambridge University Press, 2014. ISBN: 9781107087200. URL: https://books.google.co.uk/books?id=B4RxBAAAQBAJ.

[Cup89] John R. G. Cupitt. "The Design and Implementation of an Operating System in a Functional Language (Miranda)". AAIDX92465. PhD thesis. Canterbury, UK: University of Kent at Canterbury, 1989.

[Dow19a] Jonathan Dowland. *Declarative Distributed Stream Processing. PhD Stage 1 Progression Report*. 2019. URL: https://jmtd.net/log/phd/dowland_phd_stage1_progression_report.pdf.

[Dow19b] Jonathan Dowland. *PhD Poster*. 2019. URL: https://jmtd.net/phd/poster/.

[Dow20] Jonathan Dowland. *Declarative Distributed Stream Processing. PhD Stage 2 Year 1 Report*. 2020. URL: https://jmtd.net/log/phd_year_3_progression/.

[Dow21a] Jonathan Dowland. *PhD blog posts in Stage 2*. 2021. URL: https://jmtd.net/phd/stage2/.

[Dow21b] Jonathan Dowland. *StrIoT Pull Requests by Jonathan Dowland between 2019-07-01 and 2021-07-01*. 2021. URL: https://github.com/striot/striot/pulls?q=is%3Apr+created%3A2019-07-01..2021-07-01+author%3Ajmtd+.

[Dow21c] Jonathan Dowland. *Queueing theory*. 2021. URL: https://jmtd.net/log/phd/queueing_theory/.

[Faua] Taylor Fausak. *Haskell Weekly News. Issue 147*. URL: https://haskellweekly.news/issue/147.html.

[Faub] Taylor Fausak. *Haskell Weekly News. Issue 210*. URL: https://haskellweekly.news/issue/210.html.

[Fauc] Taylor Fausak. *Haskell Weekly News. Issue 215*. URL: https://haskellweekly.news/issue/215.html.

[Hir+14] Martin Hirzel et al. "A Catalog of Stream Processing Optimizations". In: *ACM Computing Surveys (CSUR)* 46 (Mar. 2014). DOI: 10.1145/2528412.

[Mor19] M. A. Morrison. *#betterposter*. 2019. URL: https://osf.io/ef53g.

[MW17] Peter Michalák and Paul Watson. "PATH2iot: A Holistic, Distributed Stream Processing System". In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2017, pp. 25–32.

[PTH01] Simon Peyton Jones, Andrew Tolmach, and Tony Hoare. "Playing by the rules: rewriting as a practical optimisation technique in GHC". In: ACM SIGPLAN. Sept. 2001. URL: https://www.microsoft.com/en-us/research/publication/playing-by-the-rules-rewriting-as-a-practical-optimisation-technique-in-ghc/.

[Sma+17] Nicholas Smallbone et al. "Quick specifications for the busy programmer". In: *Journal of Functional Programming* 27 (2017), e18. DOI: 10.1017/S0956796817000090.

[Var20] Various. *14th ACM International Conference on Distributed and Event-based Systems*. 2020. URL: https://2020.debs.org/.