PhD Proposal: Declarative, Distributed Functional Stream Processing

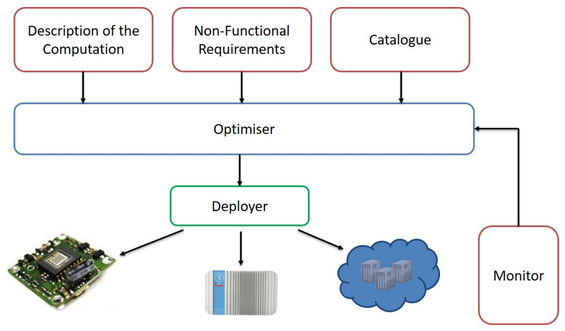
Jonathan Dowland <jon.dowland@ncl.ac.uk>, March 2018

Background

Various modern applications in domains ranging from smart cities to healthcare have a requirement for the timely processing of very large volumes of data, such as that generated by sensors in the Internet of Things (IoT). Such systems may need to meet a range of other requirements, including reliability, security, or energy efficiency, for example to prolong the battery life of sensors in the field.

The combination of requirements, very high volume of data 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, which is sent on to a local gateway (e.g. smartphone, or an embedded device) to be collated before being passed on to the Cloud for processing. However, in some domains it could be beneficial to perform some processing on the gateway or on the sensors themselves, 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.

Foundations



We are exploring an alternative approach: a system whereby the stream processing operations and the non-functional requirements are described declaratively as inputs to an Optimiser, which automatically determines the most appropriate deployment onto the available resources, which may include sensors and gateways. Monitoring of the deployment is used to evaluate the performance of the Optimiser and could also be used for run-time adaptation. The initial implementation (by Peter Michalák) used an extended version of SQL as the method of describing the computation.

In contrast, in our project we are exploring using functional programming to describe the computation. Using the pure functional language Haskell, a prototype has been developed[1] where the stream processing is defined in terms of a restricted set of functional operators with well understood semantics. This prototype has two distinct components: Library code to support stream processing in which segments of the stream are spread across multiple compute nodes; and the Optimiser.

[1] https://github.com/striot/striot

My research will be focussed on the optimiser and deployer (another PhD student — Adam Cattermole — is working on the stream processing library).

Existing progress

As well as educating myself on functional programming, my work so far has been to design and build an initial optimiser and deployment system. This has resulted in an end-to-end system that takes in a stream-processing graph and outputs source code ready for deployment, partitioned according to a user-defined mapping. We employ a library “Alga”[2] to describe the stream-processing graph. Alga is based on an algebra of graphs. This provides us with a strong formal foundation for future work on graph rewriting.

When this program is executed, a set of individual source code files are generated, one per partition. These can be deployed to separate nodes (via a system such as Docker Compose[3]) and executed.

Further work

My future work will focus on the optimisation of distributed stream processing graphs. We will explore two opportunities:

Automatic Partitioning of the Stream Processing Graph: The present system requires the user to supply a predetermined partitioning scheme, specifying which functional operators shall be distributed to each node in the deployment environment. We will explore the automatic selection of a mapping by the Optimiser based on the non-functional requirements, a “catalogue” of available infrastructure for deployment (sensors, gateways, public/private cloud nodes, etc.), and a set of cost models predicting the cost of a specific deployment. For example, it has been demonstrated that there can be benefits in performing stream processing on sensors themselves, to reduce the volume or frequency of data transmission onwards to gateways, thereby improving sensor battery life[4]. Initially, the Optimiser will generate a configuration of the input stream-processing graph once, based on the information available to it at “compile” time. However, we enhance this through collecting run time performance information in order to evaluate the performance of the Optimiser and whether or not it has made an appropriately optimal assignment of operators and partitions. There are a number of ways that this could be achieved in a black-box fashion, depending on the specifics and features provided by the deployment environment. It is likely desirable to be able to collect information on the performance of individual stream operators, perhaps amongst several deployed to a

[2] Andrey Mokhov. 2017. Algebraic graphs with class (functional pearl). In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell* (Haskell 2017). ACM, New York, NY, USA, 2-13. DOI: https://doi.org/10.1145/3122955.3122956

[3] https://docs.docker.com/compose/

[4] P. Michalák and P. Watson, "PATH2iot: A Holistic, Distributed Stream Processing System," *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Hong Kong, 2017, pp. 25-32.

particular node. To achieve this, we may wish to extend the stream graph to add “taps” for collecting such data. If the system was to be extended to operate at run-time, this performance information could be used as an input for run-time recalibration of the stream processing graph, without dropping any data in transit from sensors. Further exploration is needed before we will know if it realistic for this could be done within the scope of this PhD.

Optimisation of the Stream Processing Graph: We will explore a range of optimisations. Some will exploit the well-understood semantics of the restricted set of functional operators provided by the system. For example, two adjacent Map operators can be “fused” together, the parameter of the new Map being composition of the parameters of the original operators[5]. This process (known as deforestation) prevents the generation of intermediate list data structures. Other sources of optimisations may include those from the relational world, as well as possibly domain-specific rules, perhaps supplied by the user of the system as an additional input.

To specify and enact these optimisations, we will explore using graph rewriting rules for the Stream Processing Graph. As currently architected, the system requires the user to describe the stream-processing graph in terms of a data structure within a Haskell program. In practise this means embedding snippets of Haskell code within string literals. It would be better if the user could compose the stream-processing graph as first-class Haskell code itself. This would require an exploration of tools and language extensions, and the possible embedding of a Haskell compiler/interpreter into the Optimiser.

[5] PEYTON JONES , S., TOLMACH , A., AND HOARE , T. 2001. Playing by the rules: Rewriting as a practical optimisation technique in GHC. In Proceedings of the Haskell Workshop.