Dhalion

Self-Regulating Stream Processing in Heron

**Avrilia Floratou, Ashvin Agrawal, Bill Graham,
Sriram Rao, Karthik Ramasamy**

Yashovardhan Sharma

UNIVERSITY OF CAMBRIDGE

Motivation

Why do we need such systems?

- Explosion of real-time data analytics needs
    - Social Media, Internet of Things (Sensors), Banks, Stock Exchanges
- Many systems offer services to handle such workloads
    - Distributed
    - Can handle hardware and software failures
- Is that enough?

HERON

APACHE STORM™
Distributed • Resilient • Real-time

APACHE Spark™

samza

Problem

Problems…

- Manual tuning of configuration knobs to achieve SLOs
- Maintenance of SLOs during unpredictable load variation or performance degradation

Solution?

So what can be done?

- That's where Dhalion comes in
- Gives streaming systems the ability to **self-regulate**
- Allows systems to react and adjust dynamically to various situations
- Eases the complexity of configuring, managing and deploying such applications

Key Idea

- **Self-regulation**
- But what does that definition imply for streaming systems?
  1. Self-tuning
  2. Self-stabilising
  3. Self-healing

Self-tuning

“A self-regulating streaming system should take the specification of a streaming application as well as a policy defining the objective, and automatically tune configuration parameters to achieve the stated objective.”

Self-stabilising

“A self-regulating streaming system must react to external shocks by appropriately reconfiguring itself to guarantee stability (and SLO adherence) at all times.”

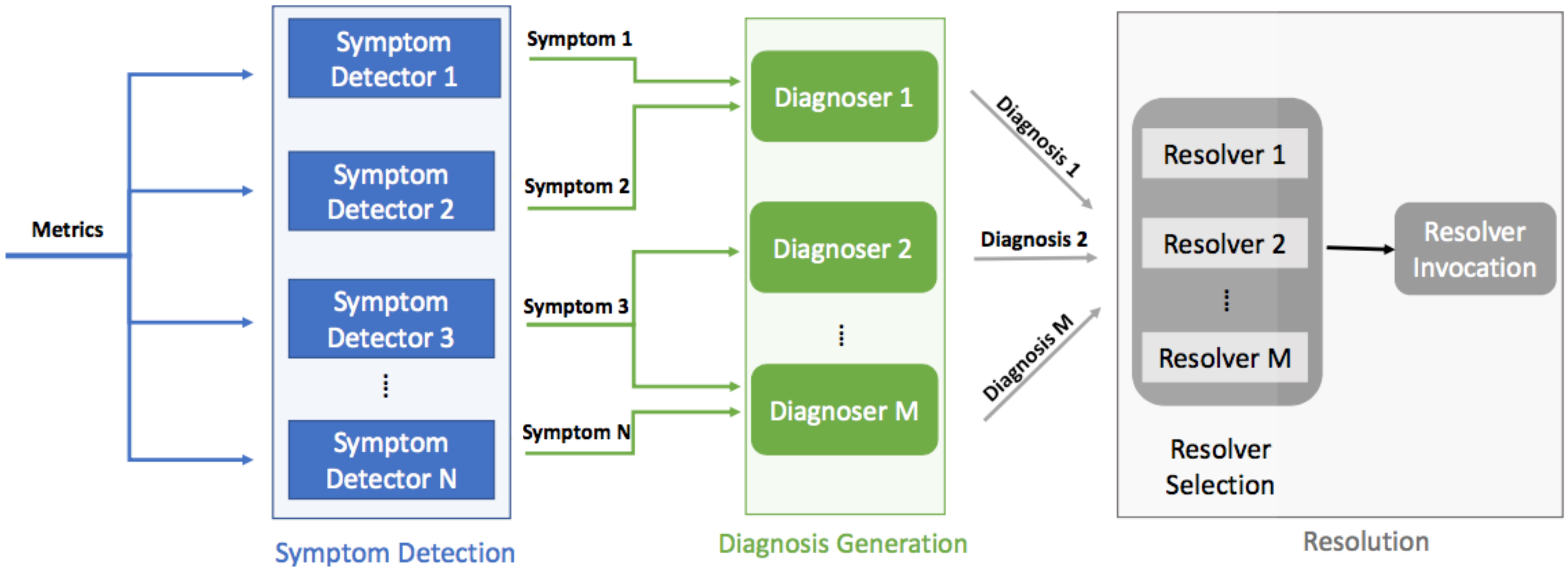
Self-healing

"A self-regulating streaming system must identify such service degradations, diagnose the internal faults that are at their root, and perform the necessary actions to recover from them."

How does it work?

- Dhalion sits on top of other frameworks
- Periodically invokes a well-defined policy
- Policy examines the status of the application and detects potential problems
- Attempts to resolve them by performing the appropriate actions

Overview

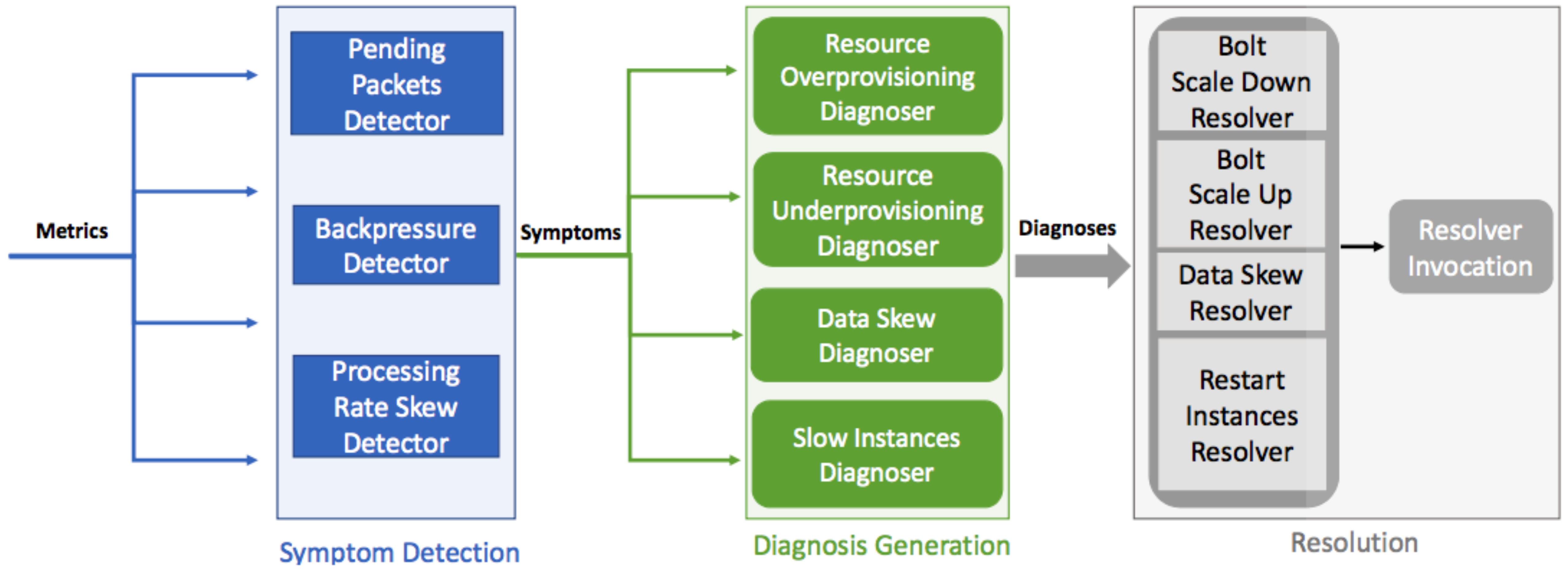

Metrics
Symptom Detector 1
Symptom Detector 2
Symptom Detector 3
⋮
Symptom Detector N
Symptom Detection
Symptom 1
Symptom 2
Symptom 3
Symptom N
Diagnoser 1
Diagnoser 2
⋮
Diagnoser M
Diagnosis Generation
Diagnosis 1
Diagnosis 2
Diagnosis M
Resolver 1
Resolver 2
⋮
Resolver M
Resolver Selection
Resolver Invocation
Resolution

Example Policy

Dynamic Resource Provisioning

“Dynamic Resource Provisioning is a policy that observes the system behaviour and dynamically provisions the topology resources so that the overall throughput is maximised while at the same time the resources are not underutilised.”

Dynamic Resource Provisioning



Experimental Evaluation

- Dhalion works well for multi-stage topologies where backpressure propagates from one stage to the other
- System is able to dynamically adjust resources according to the load, and maximise throughput
- System is able to automatically reconfigure a topology to meet SLOs
- Dhalion's actions are unaffected by noise and transient changes
- Can bring the topology to healthy state even when multiple problems occur

Summary

- Introduction to the notion of self-regulating streaming systems
- **Dhalion** : A modular and extensible system deployed on top of streaming systems
- Provides **self-regulating** capabilities through the the execution of various policies
- Allows users to define their own policies and incorporate them into their streaming applications

Critique

- The flaws in the Blacklisting mechanism
- Binary attribution of symptoms to causes by Diagnosers
- Categorising backpressure using a threshold

Questions?

References

1. Floratou, Avrilia, et al. "Dhalion: self-regulating stream processing in heron." Proceedings of the VLDB Endowment 10.12 (2017): 1825-1836.

2. S. a. Kulkarni. Twitter Heron: Stream Processing at Scale. In ACM SIGMOD '15, pages 239–250, 2015.

3. Heron Code Repository. https://github.com/twitter/heron.

4. Spark Streaming. http://spark.apache.org/streaming/.

5. Apache Samza. http://samza.apache.org/

6. Apache Aurora. http://aurora.apache.org/.

7. Apache Flink. https://flink.apache.org/.

8. Apache Kafka. http://kafka.apache.org/.