Minimizing Self-Adaptation Overhead in Parallel Stream Processing for Multi-Cores

Adriano Vogel[1], Dalvan Griebler[1], Marco Danelutto[2] and Luiz Gustavo Fernandes[1]

PUCRS

[1] Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil
[2] Department of Computer Science, University of Pisa (UNIPI), Italy

IN SUPREMÆ DIGNITATIS 1343

2019

Outline

- Introduction
- Overview of SPar
- Problem
- Related Work
- Solution
- Evaluation
- Conclusion
- References

Introduction

- **Stream Processing Applications [1] [2]**

Introduction

- **Challenges**
  - Parallel Programming complexities
  - Productivity
  - Programming and system architecture expertises
- **High-level parallel programming frameworks**
  - Intel Threading Building Blocks (TBB)
  - FastFlow
  - StreamIt
- **DSL (Domain-Specific Language)**
  - SPar

Overview of SPar

SPar
Stream . Parallelism

- SPar - a DSL for Stream Parallelism [7]

```
[[spar::ToStream]] while(true){
   item = read();
    [[spar::Stage,spar::Input(item),spar::Output(item),spar::Replicate(N)]]{
      item = filter(item);
    }
    [[spar::Stage,spar::Input(item)]]{
      write(item);
    }
}
```

ID
AUX

attribute specifier sequence (since C++11)

Introduces implementation-defined attributes for types, objects, code, etc.

`[[attr]] [[attr1, attr2, attr3(args)]] [[namespace::attr(args)]] alignas_specifier`

Formally, the syntax is

`[[ attribute-list ]]` (since C++11)

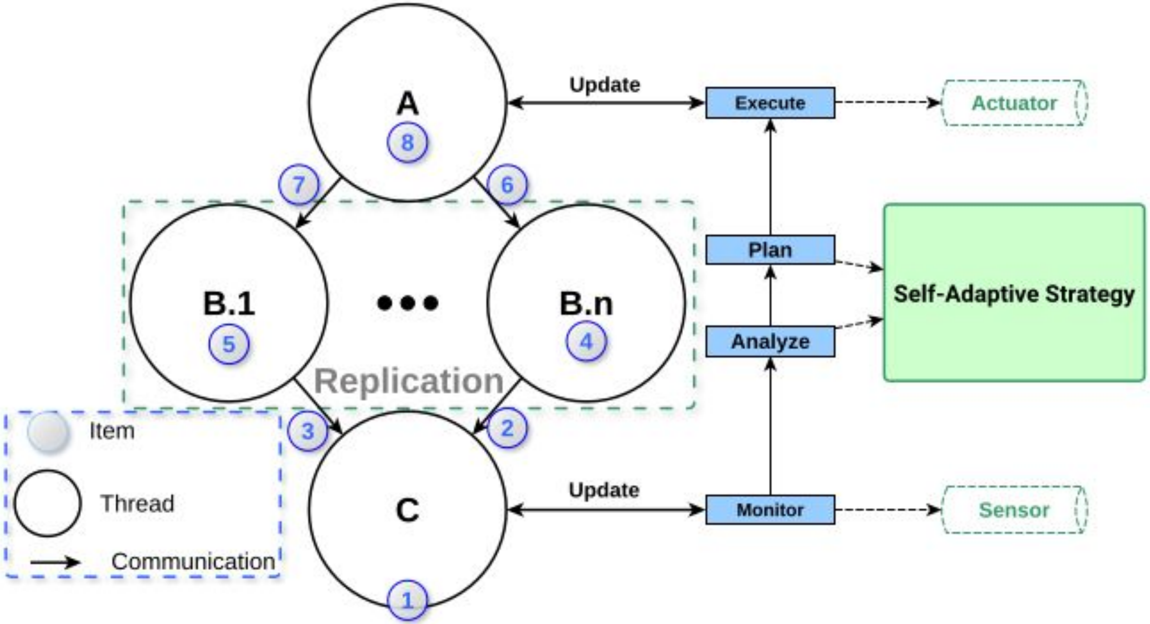
Overview of SPar

- SPar - a DSL for Stream Parallelism [10]

```
[[spar::ToStream]] while(true){
  item = read();
  [[spar::Stage,spar::Input(item),spar::Output(item),spar::Replicate(N)]]{
    item = filter(item);
  }
  [[spar::Stage,spar::Input(item)]]{
    write(item);
  }
}
```

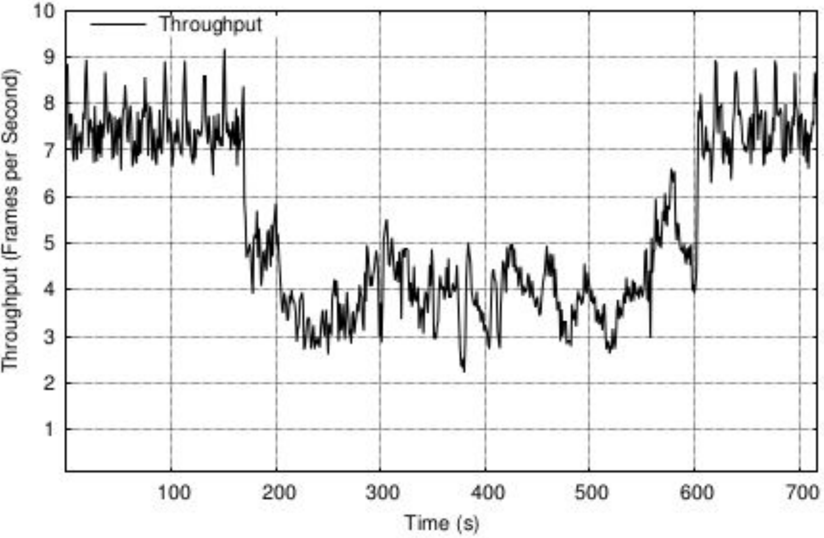
Overview of SPar

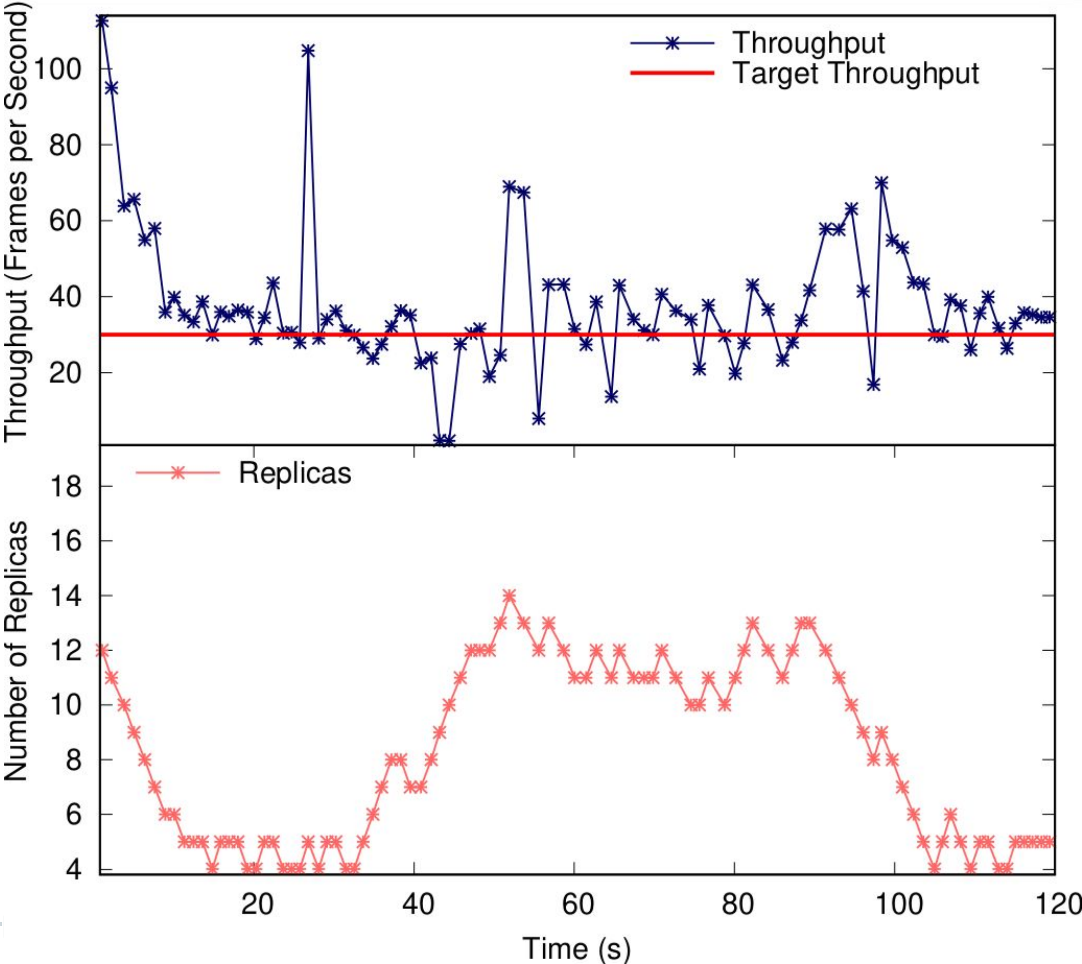
- Self-adaptive parallelism in SPar [2,3]



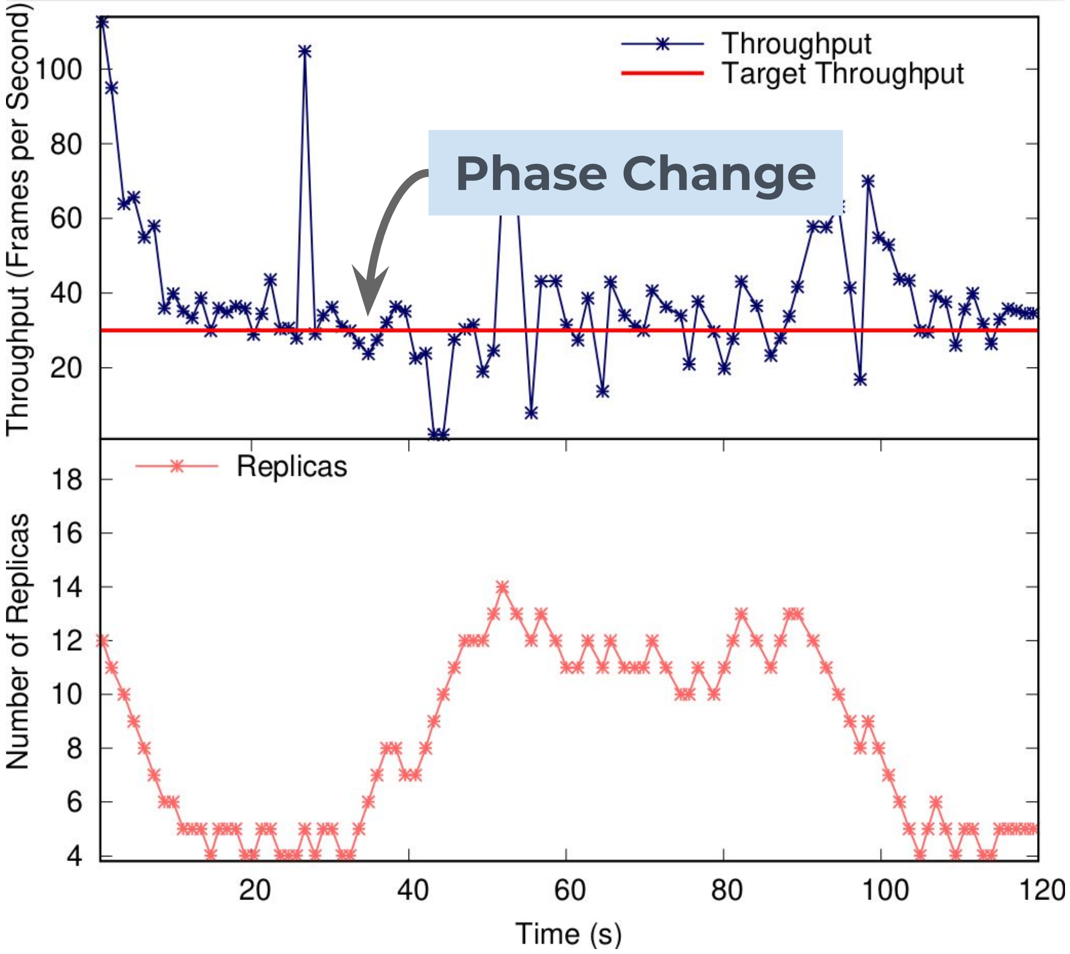
Problem

Video App - Lane Detection

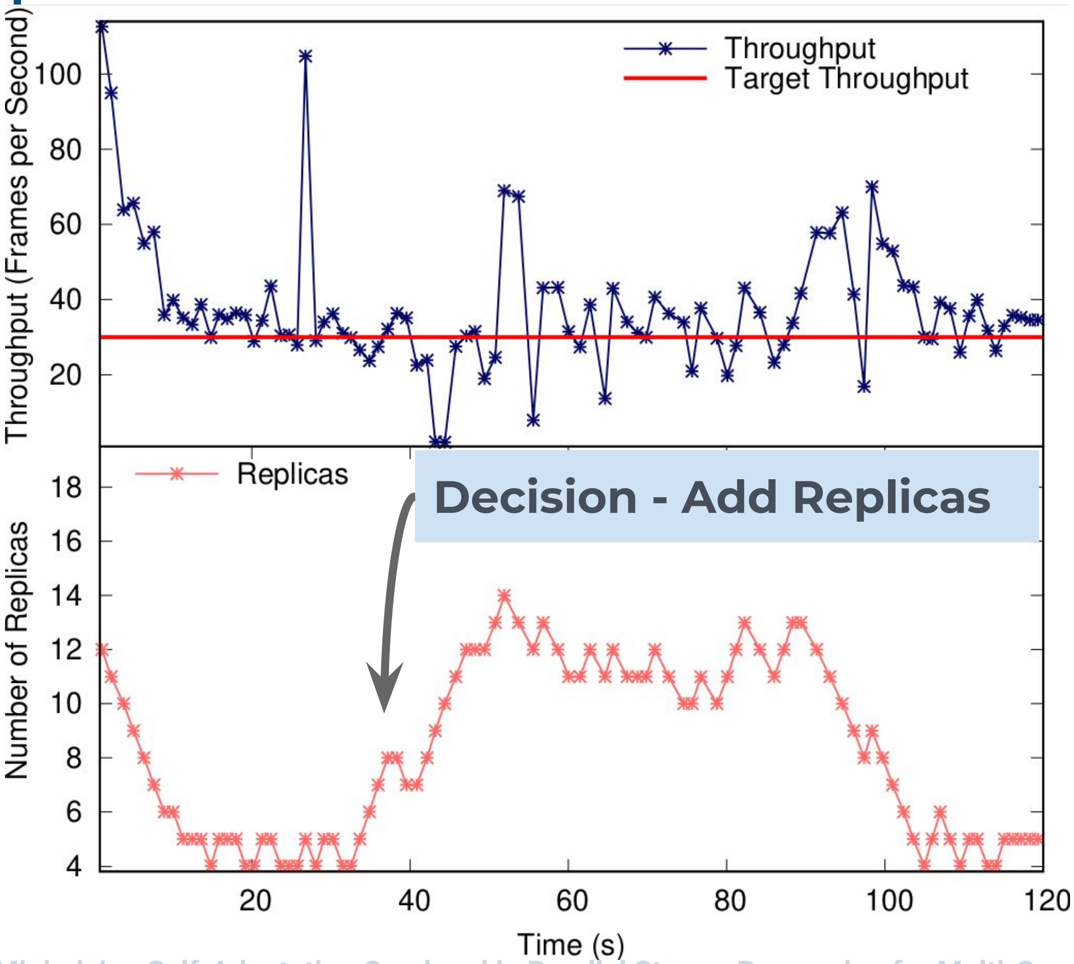




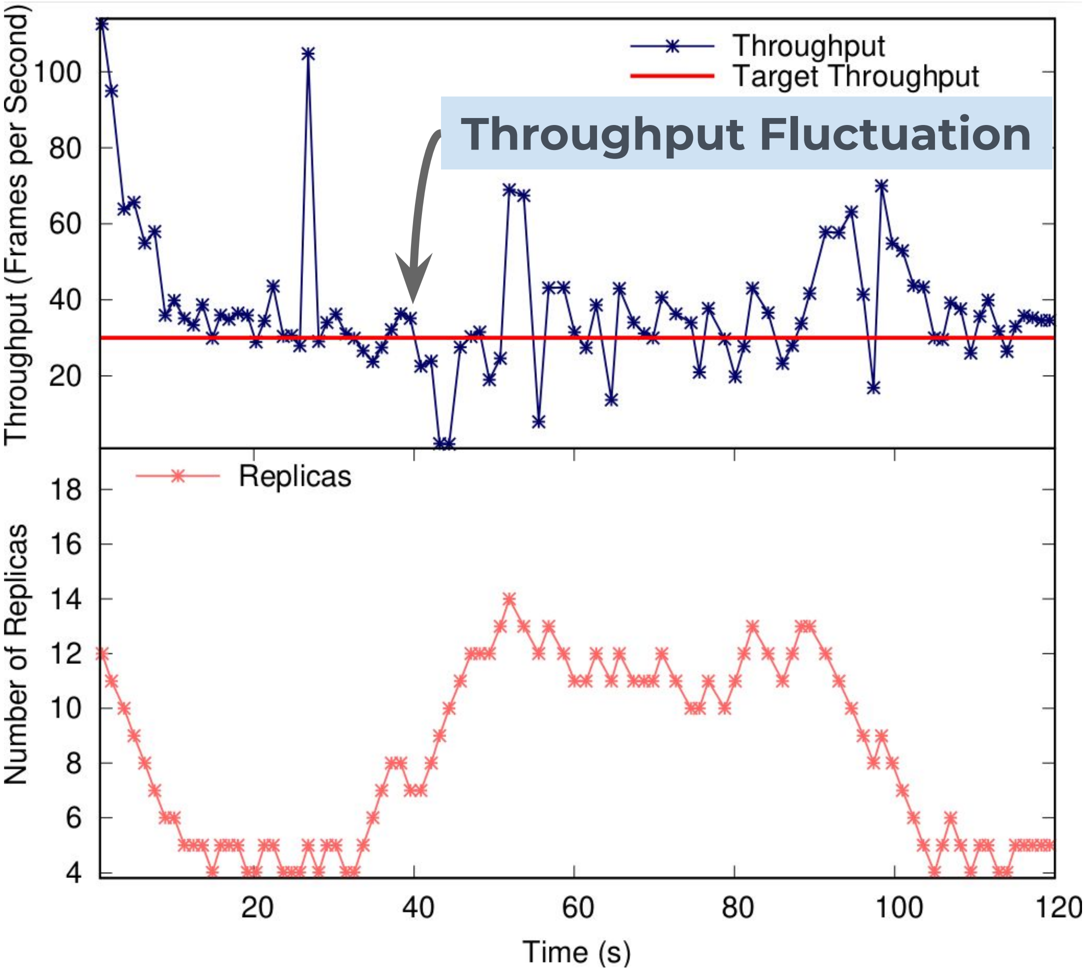
Problem


Throughput
Target Throughput
Phase Change
Throughput (Frames per Second)
100
80
60
40
20
Replicas
Number of Replicas
18
16
14
12
10
8
6
4
20
40
60
80
100
120
Time (s)

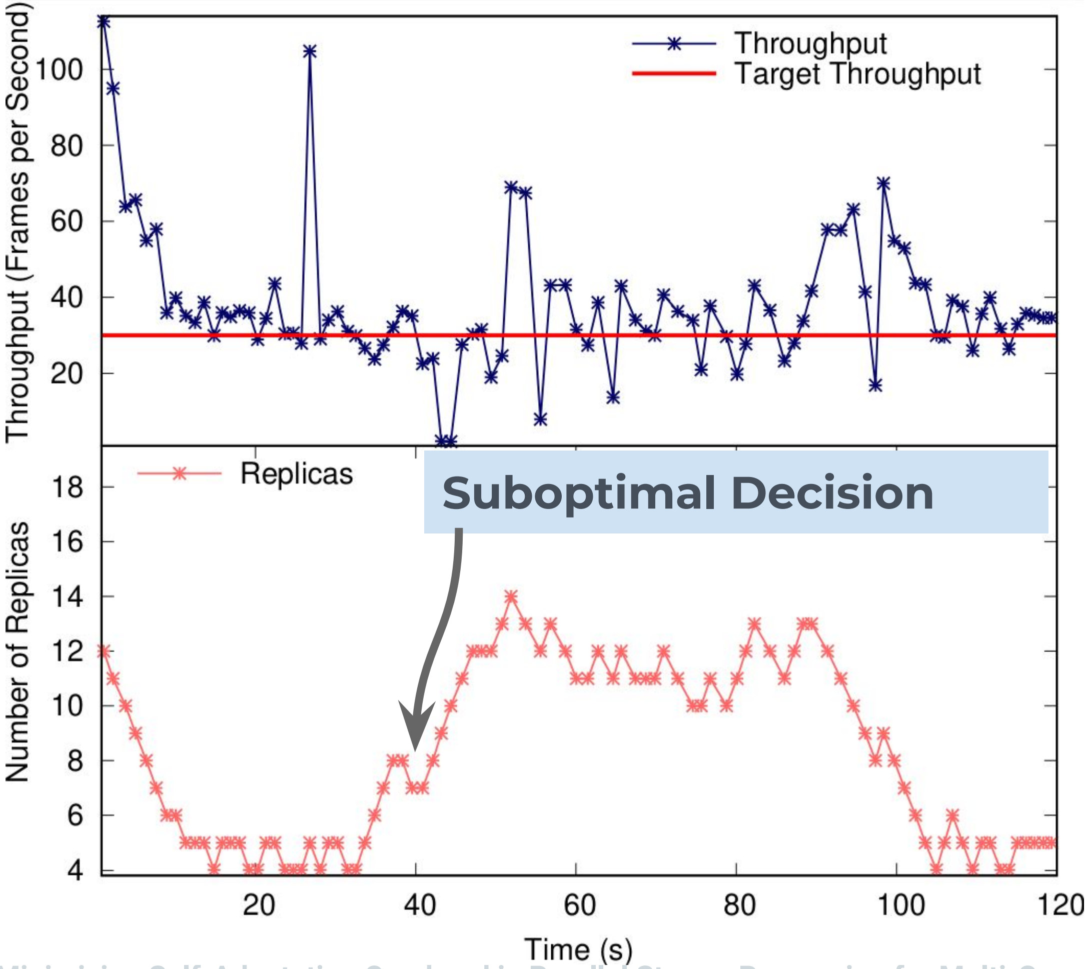
Problem


Throughput
Target Throughput
Throughput (Frames per Second)
100
80
60
40
20
Replicas
Decision - Add Replicas
Number of Replicas
18
16
14
12
10
8
6
4
20
40
60
80
100
120
Time (s)

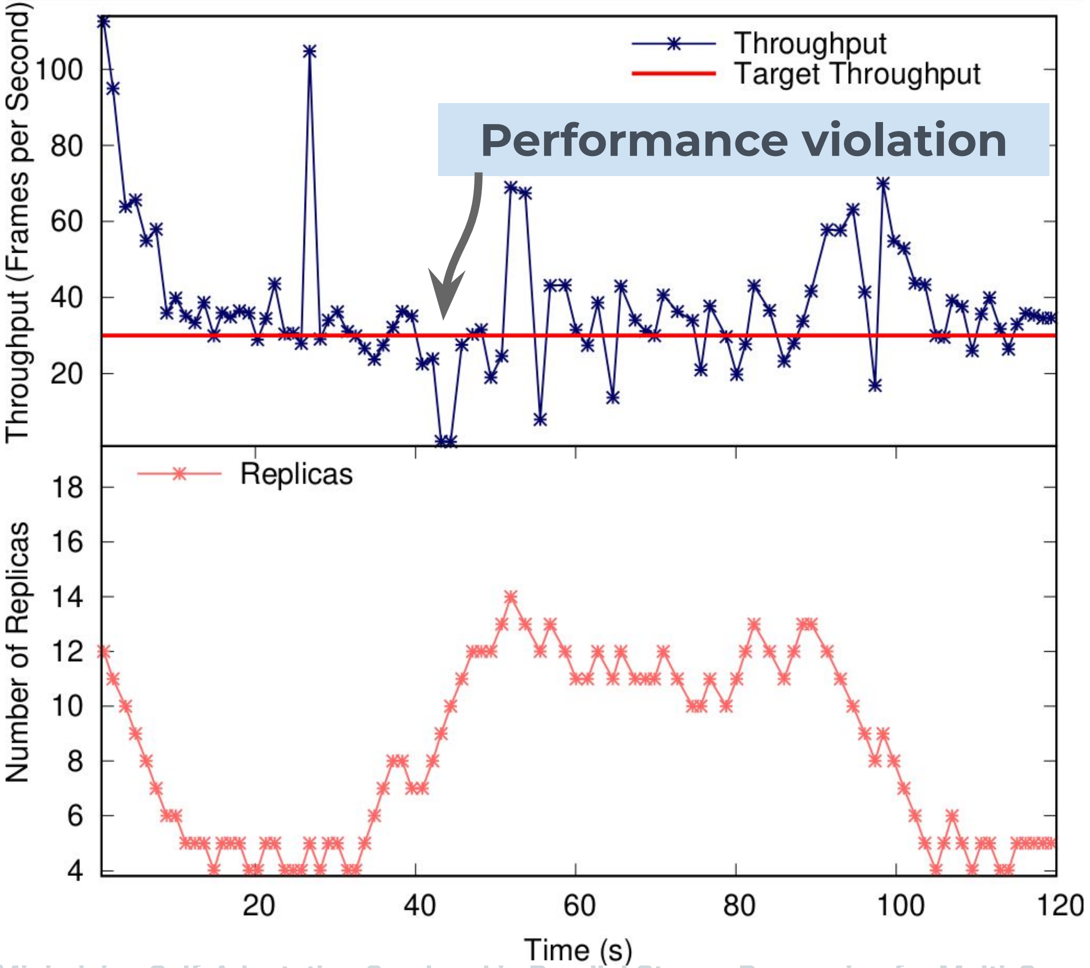
Problem


Throughput
Target Throughput
Throughput Fluctuation
Throughput (Frames per Second)
100
80
60
40
20
Replicas
Number of Replicas
18
16
14
12
10
8
6
4
20
40
60
80
100
120
Time (s)

Problem

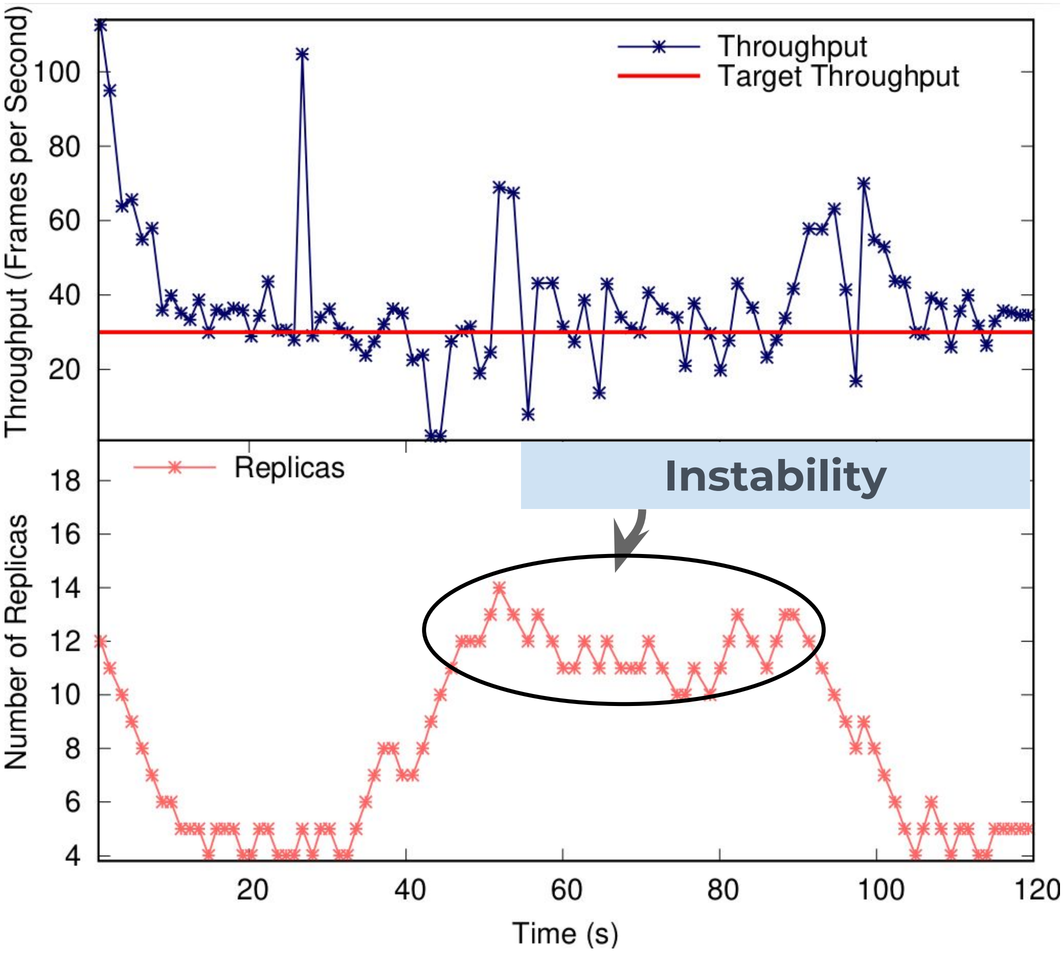

Throughput (Frames per Second)
100
80
60
40
20
Throughput
Target Throughput
Replicas
Suboptimal Decision
18
16
14
12
10
8
6
4
Number of Replicas
20
40
60
80
100
120
Time (s)

Problem

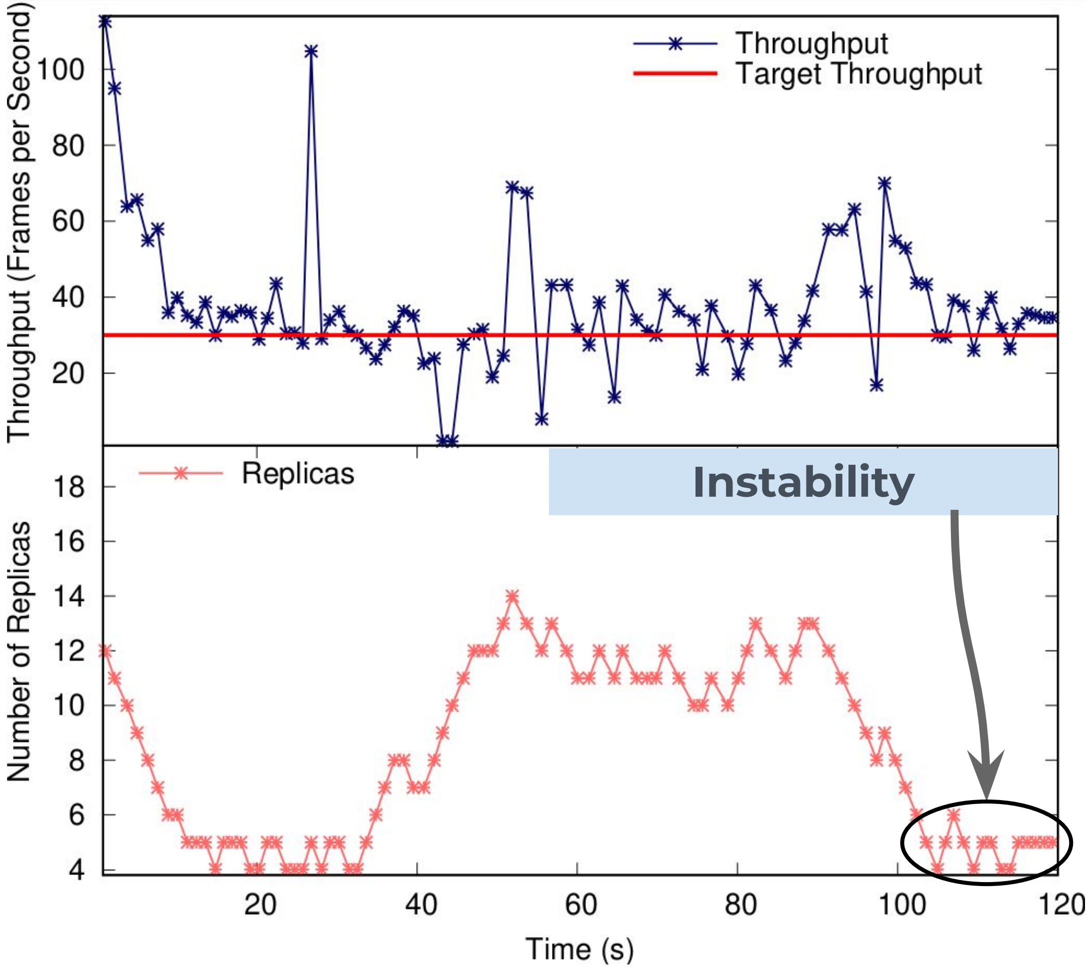


Throughput
Target Throughput
Performance violation
Throughput (Frames per Second)
100
80
60
40
20
Replicas
Number of Replicas
18
16
14
12
10
8
6
4
20
40
60
80
100
120
Time (s)

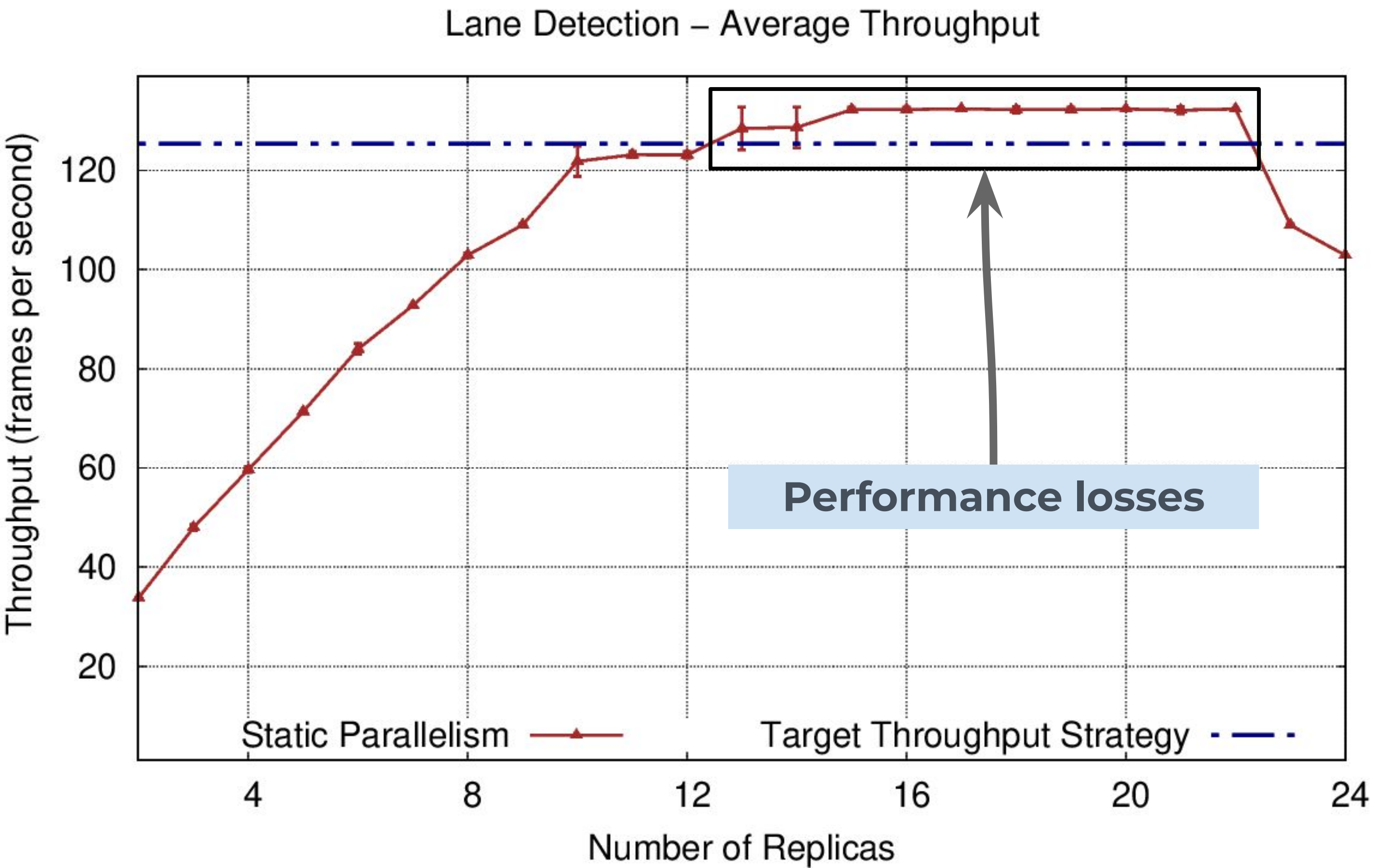
Problem


Throughput (Frames per Second)
100
80
60
40
20
Throughput
Target Throughput
Number of Replicas
18
16
14
12
10
8
6
4
Replicas
Instability
20
40
60
80
100
120
Time (s)

Problem


Throughput
Target Throughput
Throughput (Frames per Second)
100
80
60
40
20
Replicas
Instability
Number of Replicas
18
16
14
12
10
8
6
4
20
40
60
80
100
120
Time (s)

Problem



Lane Detection – Average Throughput
Throughput (frames per second)
120
100
80
60
40
20
4
8
12
16
20
24
Number of Replicas
Static Parallelism
Target Throughput Strategy
Performance losses

Targets

- Goals:
  - Improve existing abstractions
  - Reduce self-adaptation overhead
  - Improve applications performance
- Contributions:
  - A new optimized strategy for self-adapting the parallelism when the programmer/user sets a target performance.
  - A comprehensive validation of our solution for parallel programming abstractions

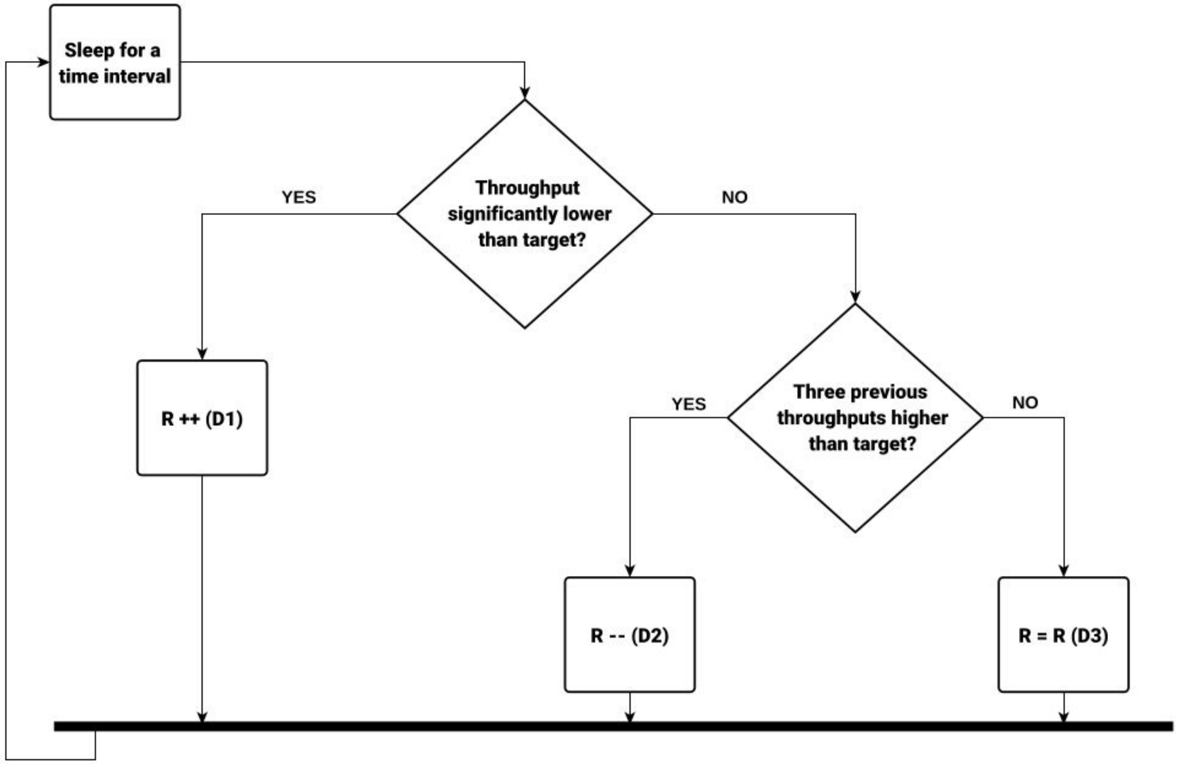
Related Work

Work	Library/System	Environment	Objective
De Sensi *et al.* [4]	NORNIR	Multi-core	Manage throughput and power consumption
De Matteis *et al.* [5]	FastFlow	Multi-core	Latency and energy efficiency
Gedik *et al.* [6]	SPL	Multi-core	High throughput without wasting computational resources
Selva *et al.* [8]	StreamIt	Multi-core	Throughput
This work	SPar	Multi-core	Improve self-adaptation for parallelism abstractions

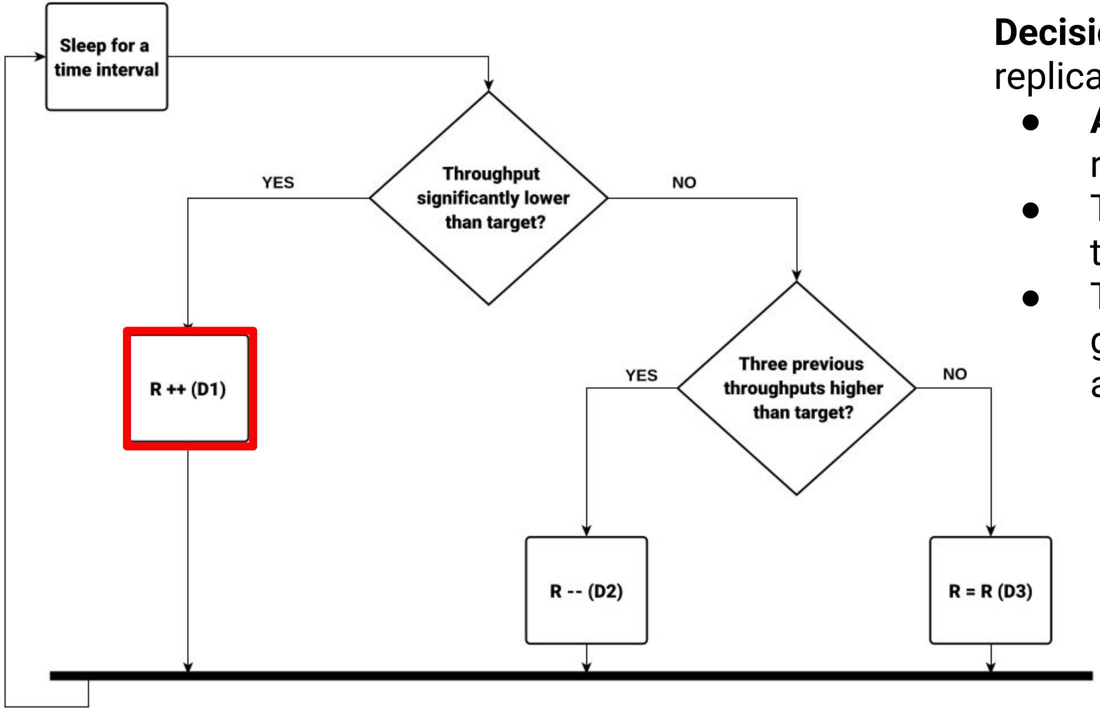
Solution

- Increase
  - Stability
  - Performance
- Reduce:
  - Setting times
- Avoid:
  - overshooting

Solution


Sleep for a time interval
Throughput significantly lower than target?
YES
NO
R ++ (D1)
Three previous throughputs higher than target?
YES
NO
R -- (D2)
R = R (D3)

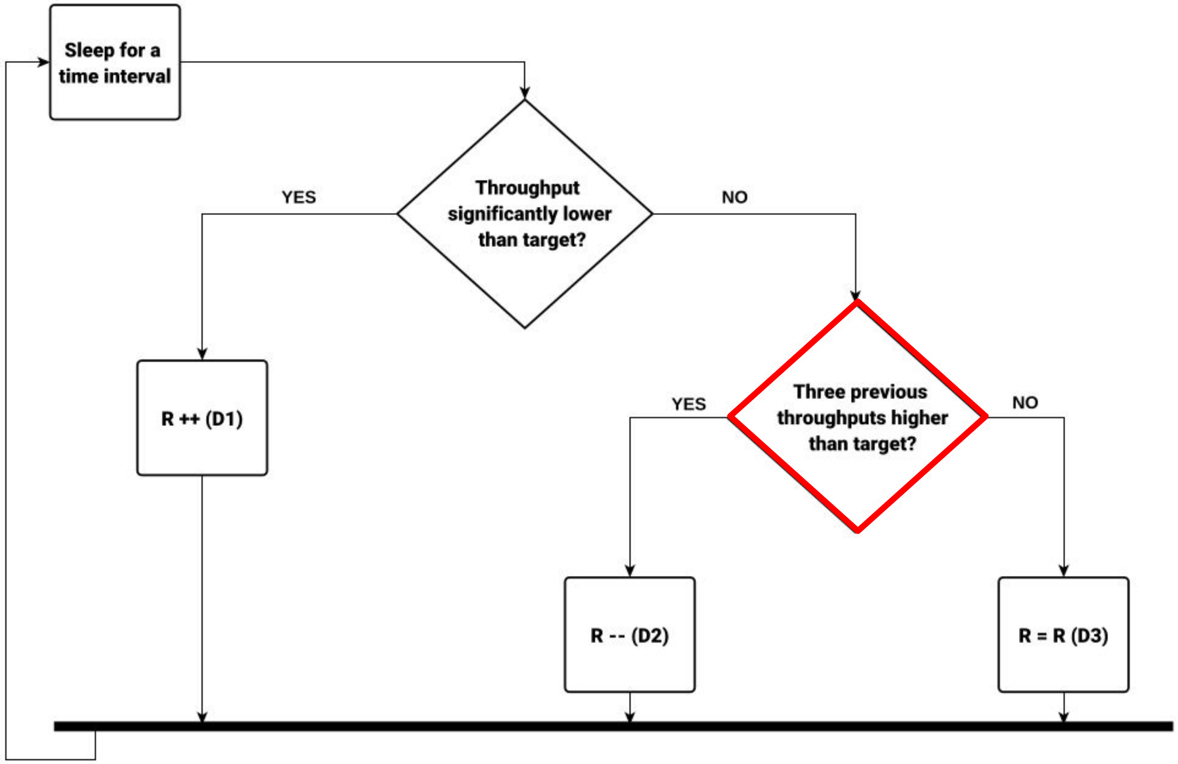
Solution



Decision 1 (D1) increases the number of replicas.

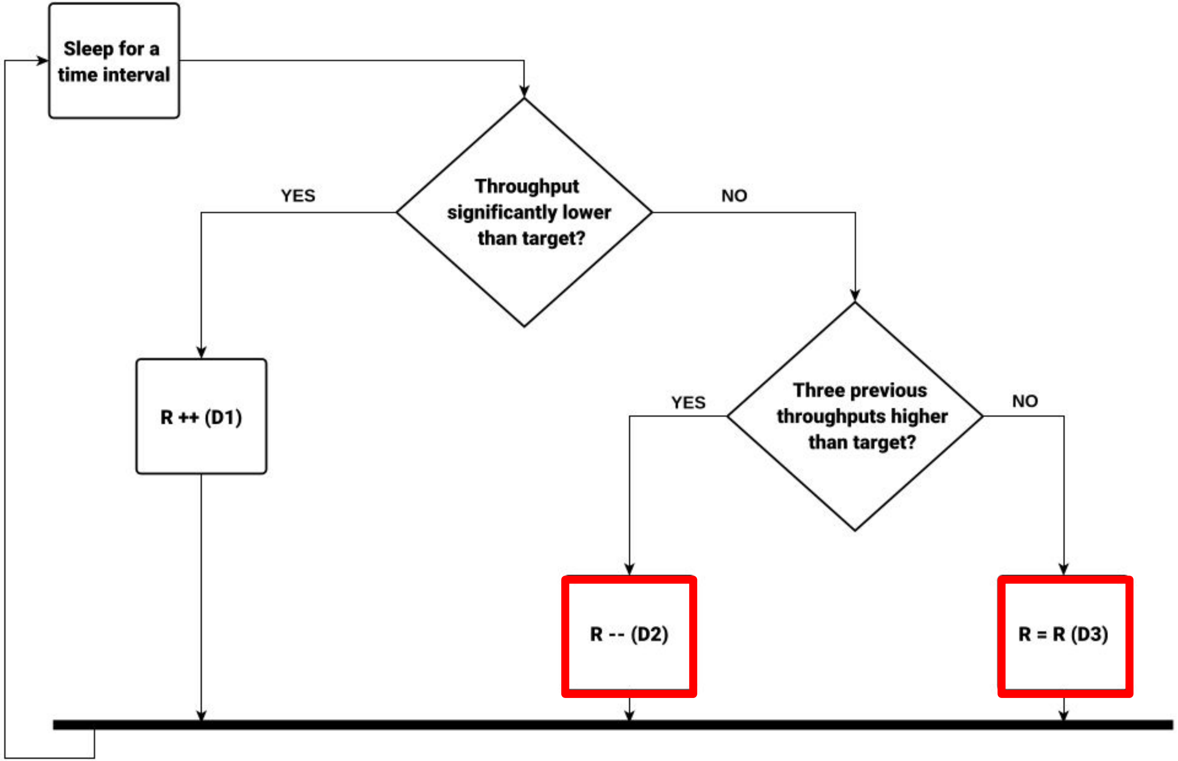
- **Adaptive number of replicas,** several replicas may be added in one step;
- The difference between actual and the target performance;
- The relation between the performance gap and the amount of resources available

Solution

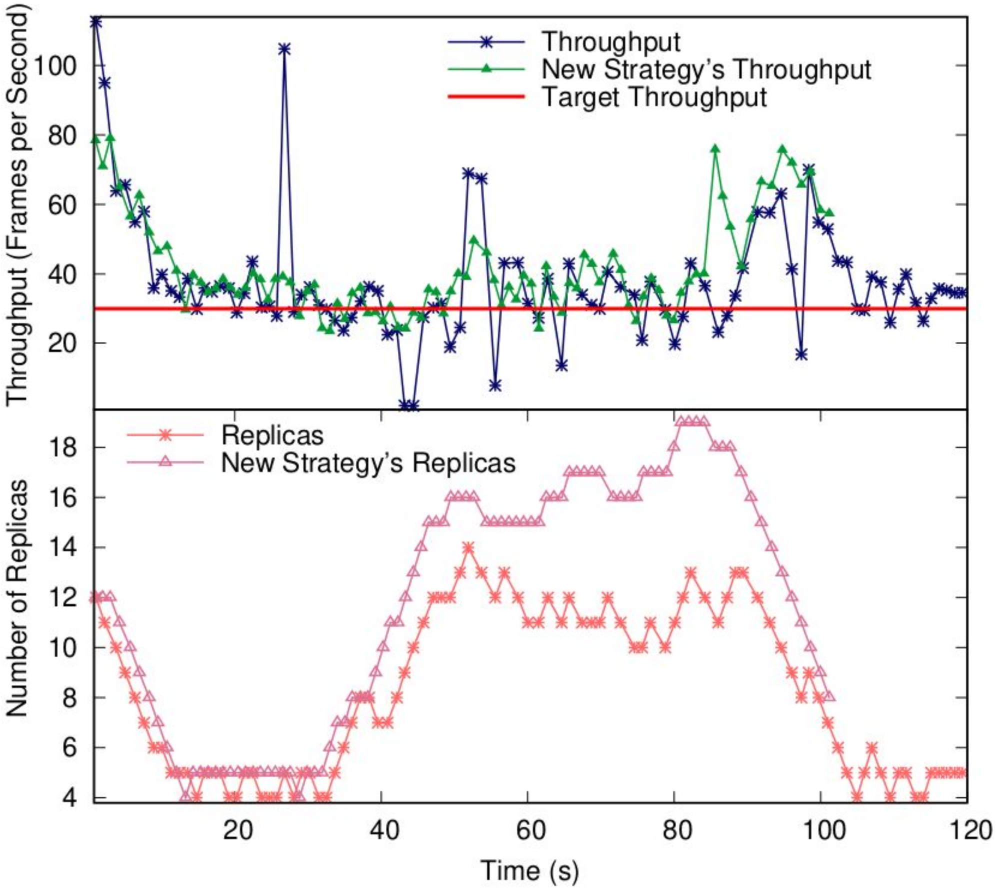


Sleep for a time interval
Throughput significantly lower than target?
YES
NO
R ++ (D1)
YES
Three previous throughputs higher than target?
NO
R -- (D2)
R = R (D3)

Solution

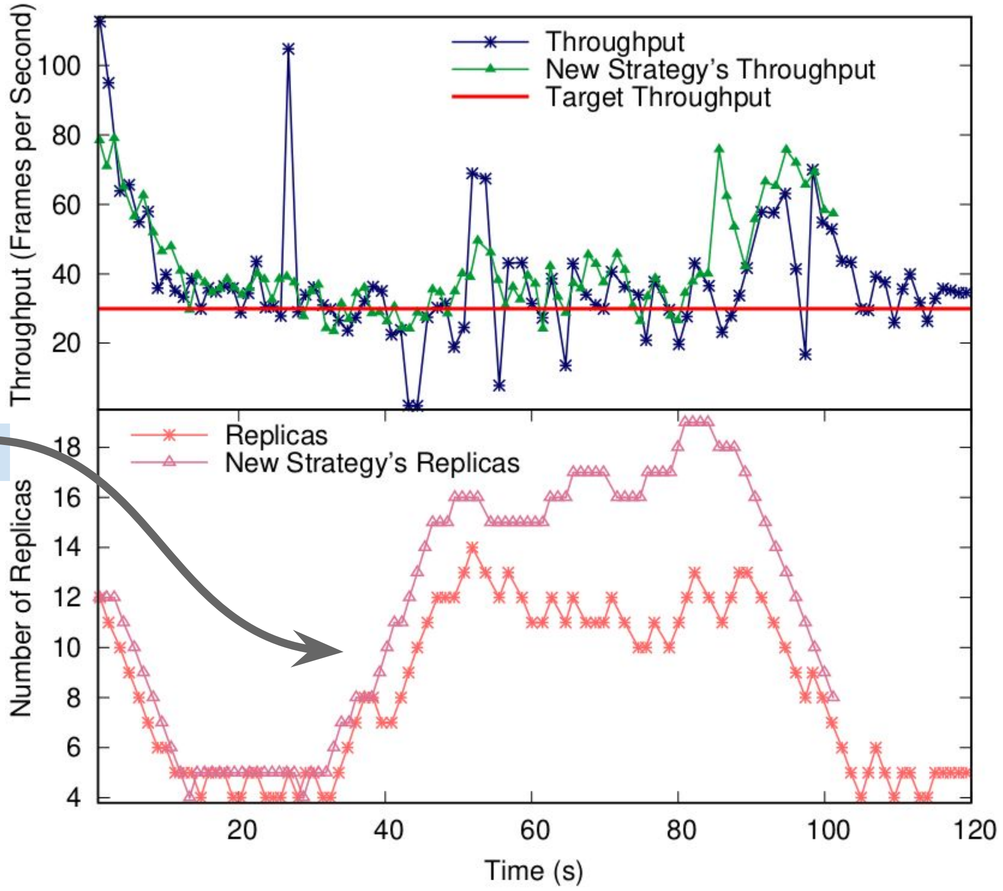

Sleep for a time interval
Throughput significantly lower than target?
YES
NO
R ++ (D1)
Three previous throughputs higher than target?
YES
NO
R -- (D2)
R = R (D3)

Characterization - Lane Detection

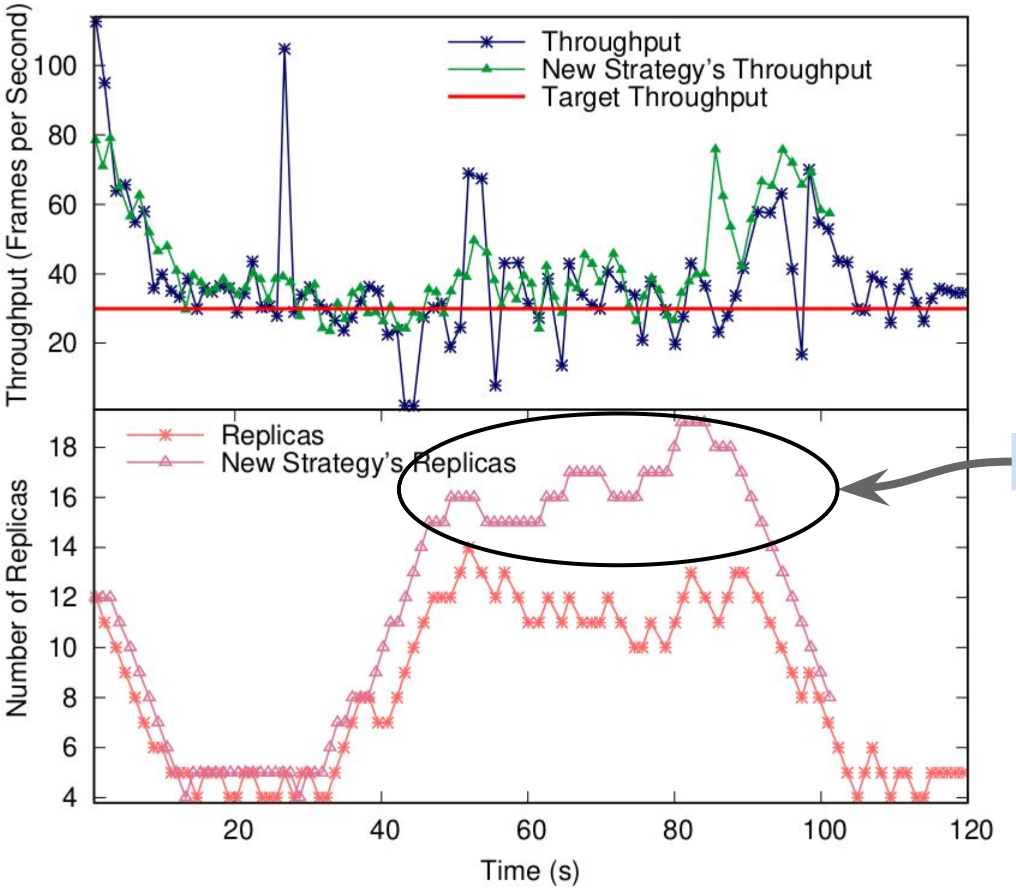


Characterization - Lane Detection

Better responsiveness

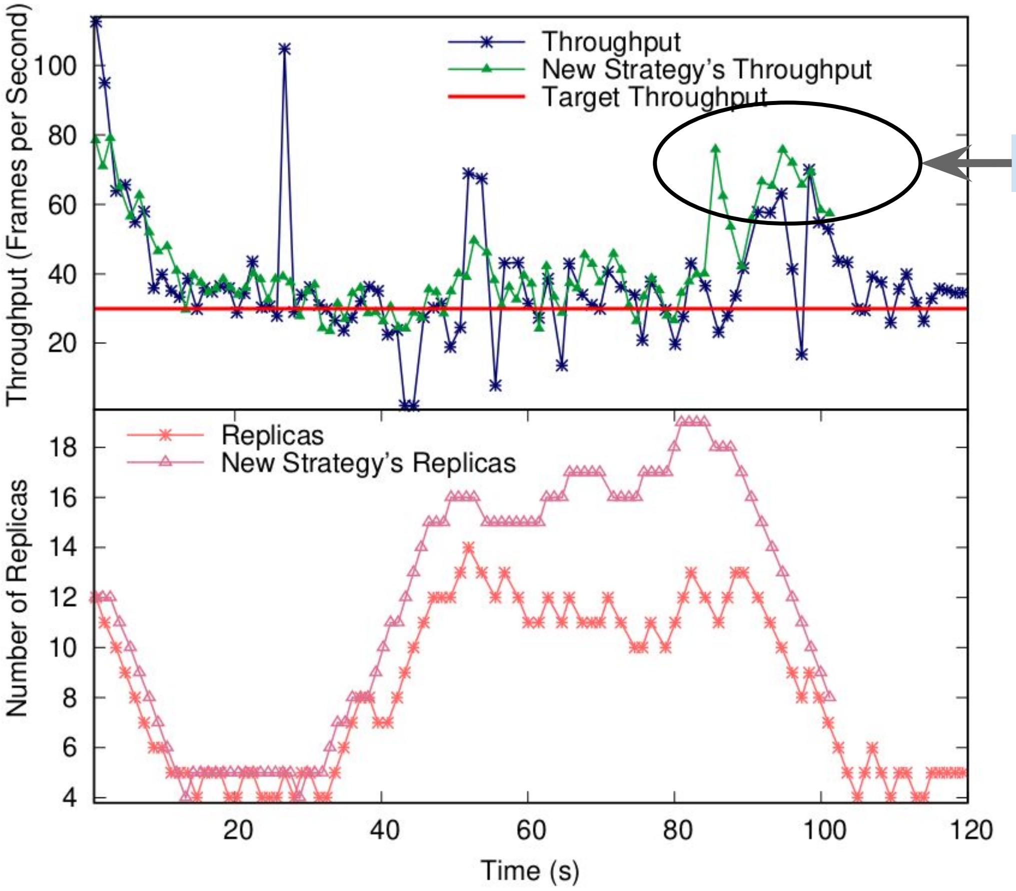


Characterization - Lane Detection



Stabler

Characterization - Lane Detection

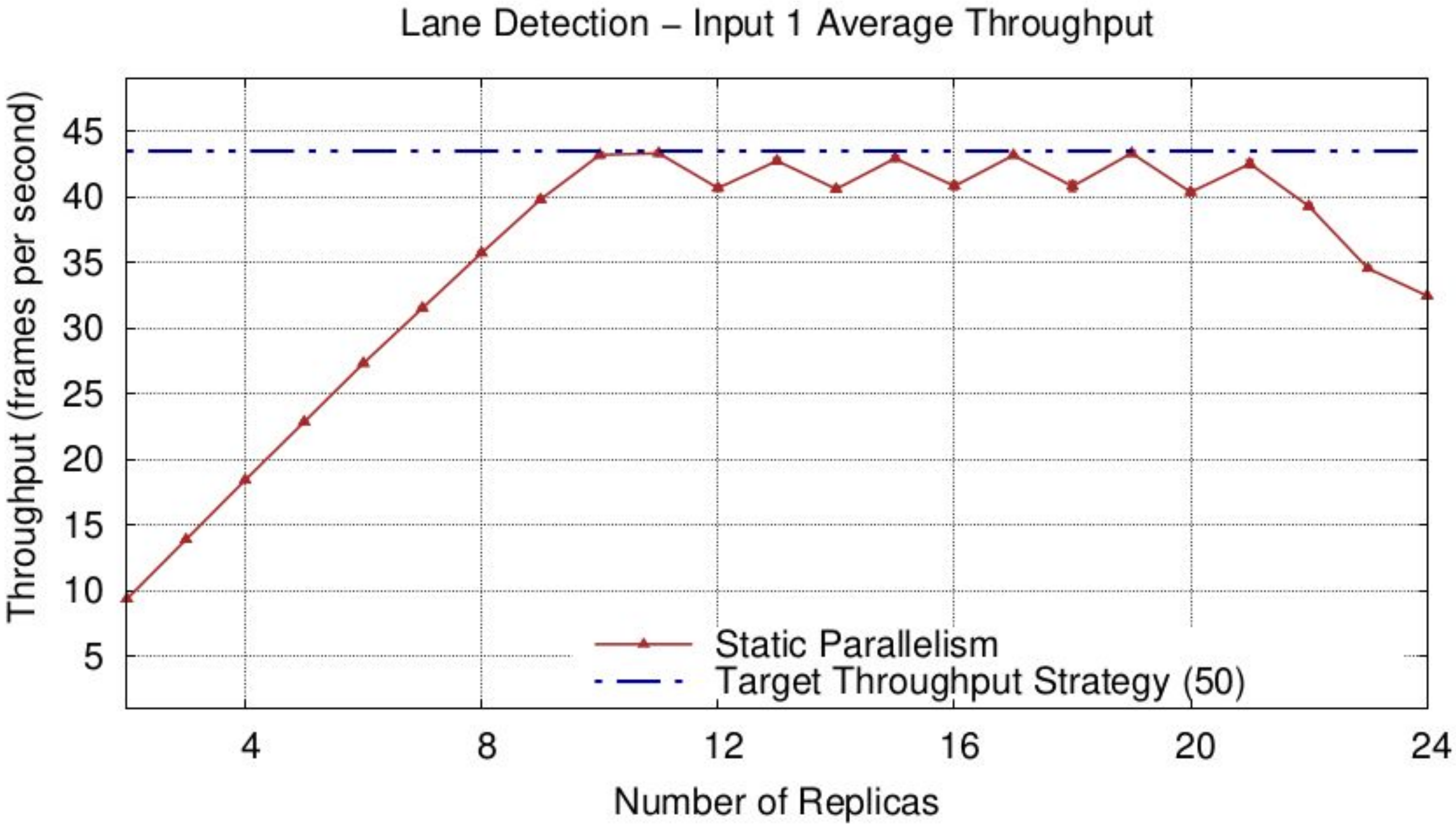


Better performance

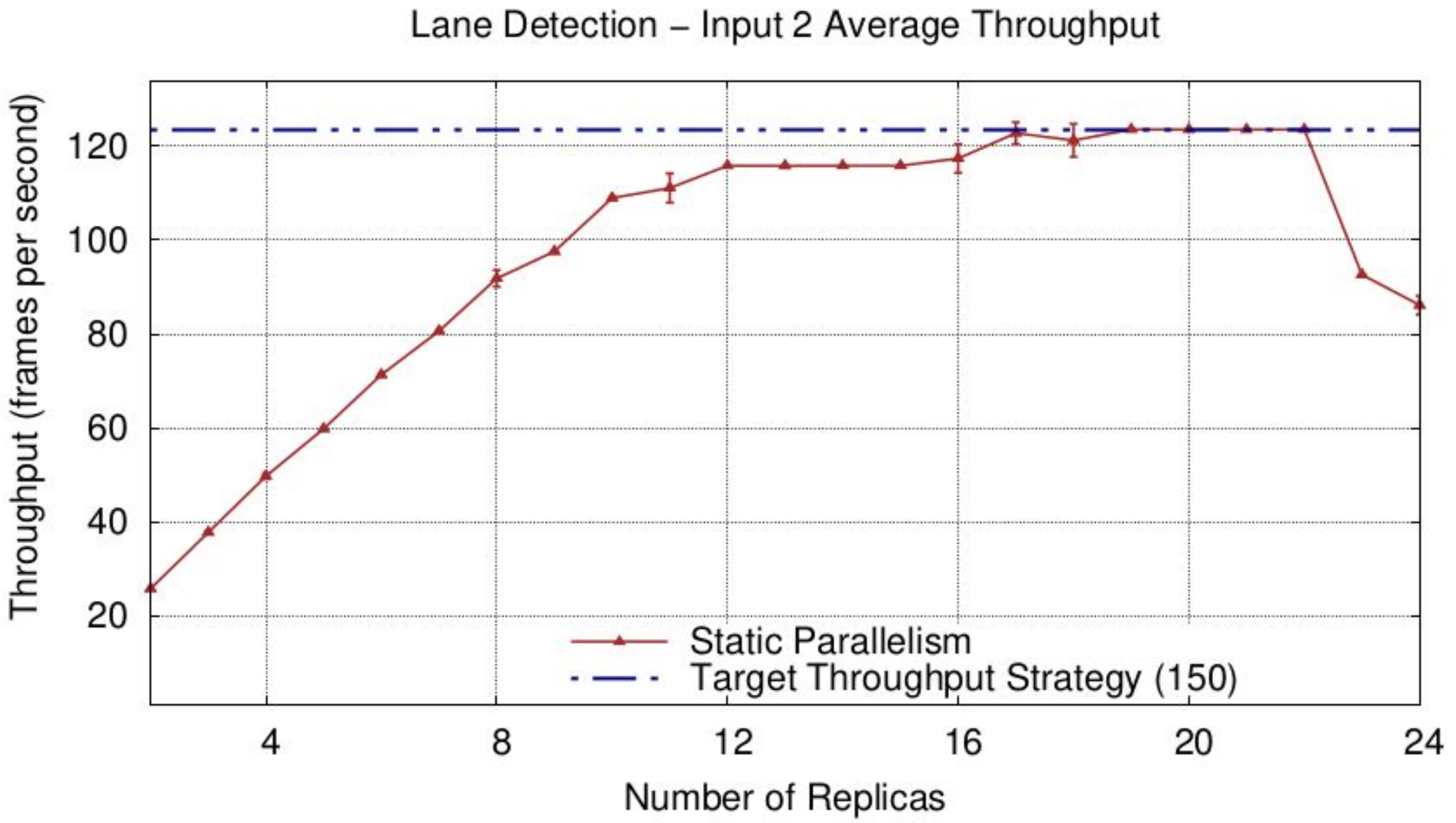
Performance Evaluation

- Multi-core machine
  - 32 GB - 2133 MHz
  - Dual-socket Intel(R) Xeon(R) CPU 2.40GHz (12 cores-24 threads).
  - Ubuntu Server 16.04 OS
  - G++ v. 5.4.0 -O3 flag
  - Ondemand scheduling
- Tested with two applications w.r.t. performance and memory consumption
- Self-adaptivity compared to static executions (fixed number of replicas)

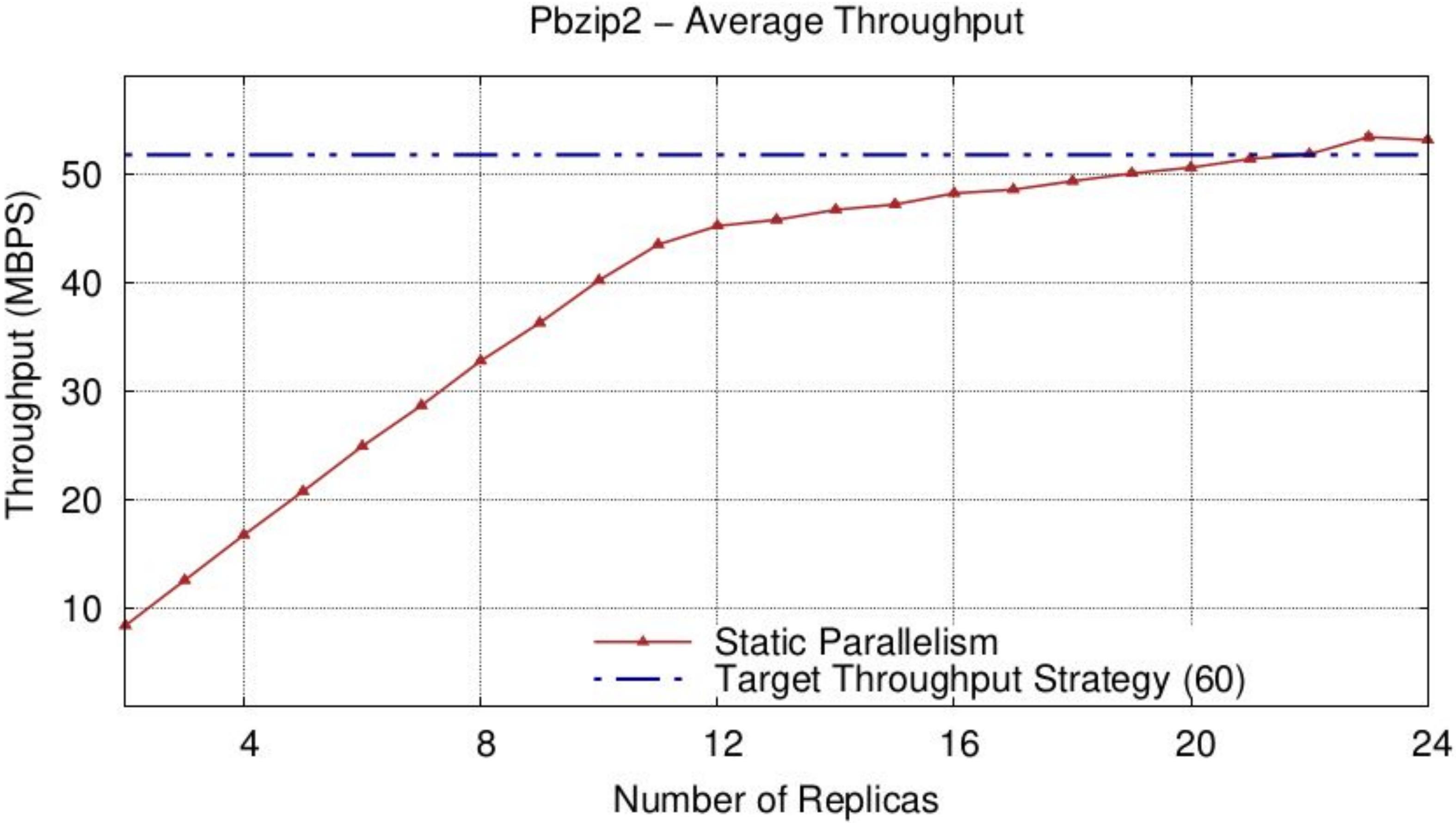
Performance



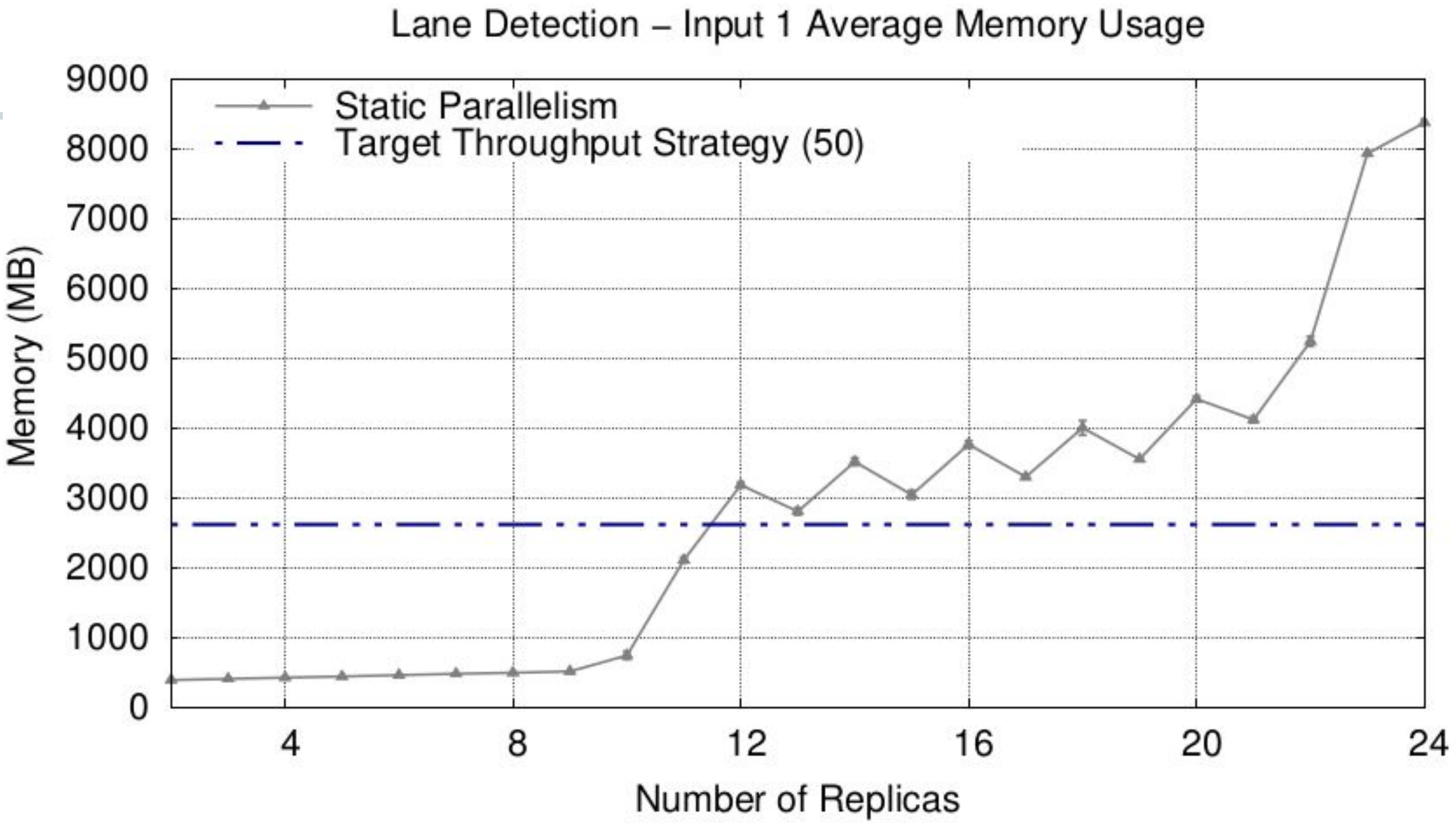
Performance



Performance



Memory consumption



Conclusion

- Implications
  - Self-adaptivity with a competitive performance
  - Low or no overhead of adaptivity
- Limitations
  - Performance may vary
  - Performance vs resources
- Future Work:
  - Extend this work to consider applications with a more complex structure
  - Use proactive approaches

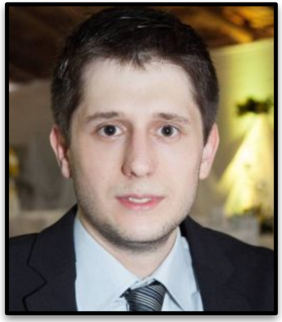
References

[1] Andrade, H.; Gedik, B.; Turaga, D. "Fundamentals of Stream Processing: Application Design, Systems, and Analytics". Cambridge University Press, 2014.
[2] A. Vogel, D. Griebler, D. D. Sensi, M. Danelutto, and L. G. Fernandes, "Autonomic and Latency-Aware Degree of Parallelism Management in SPar," in Euro-Par 2018:Parallel Processing Workshops. Turin, Italy: Springer, August 2018, p. 12.
[3] D. Griebler, A. Vogel, D. De Sensi, M. Danelutto, and L. G. Fernandes, "Simplifying and implementing service level objectives for stream parallelism," The Journal of Supercomputing, Jun 2019.
[4] Sensi, D. D.; Torquati, M.; Danelutto, M. "A reconfiguration algorithm for power-aware parallel applications", ACM Transactions on Architecture and Code Optimization (TACO), vol. 13–4, 2016, pp. 43.
[5] De Matteis, T.; Mencagli, G. "Keep calm and react with foresight: strategies for low-latency and energy-efficient elastic data stream processing". In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2016, pp. 13.
[6] Gedik, B.; Schneider, S.; Hirzel, M.; Wu, K.-L. "Elastic scaling for data stream processing", IEEE Transactions on Parallel and Distributed Systems, vol. 25–6, 2014, pp. 1447–1463.
[7] Griebler, D. "Domain-Specific Language & Support Tool for High-Level Stream Parallelism", Ph.D. Thesis, Faculdade de Informática - PPGCC - PUCRS, Porto Alegre, Brazil, 2016, 243p.
[8] Selva, M.; Morel, L.; Marquet, K.; Frenot, S. "A monitoring system for runtime adaptations of streaming applications". In: Parallel, Distributed and Network Based Processing (PDP), 2015 23rd Euromicro International Conference on, 2015, pp. 27–34.

EURO-PAR CONFERENCE 2019

Thank you!

E-mail: adriano.vogel@edu.pucrs.br



Adriano Vogel

Dalvan Griebler

Marco Danelutto

Luiz Gustavo Fernandes