Evaluating Approaches to Detect Bottlenecks in the Pipe & Filter Framework TeeTime

Adrian Pegler

March 18, 2016
1. Introduction

2. Foundation

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
Studies

- Kissmetrics\(^a\):
  - Hypothetical web-application with daily sales of 100,000$
    → 7% annually sales loss
  - Amazon: 100ms page delay → 1% decrease in sales [3]
  - Google: 500ms page delay → 20% drop in traffic [3]

\(^a\)http://blog.kissmetrics.com/loading-time/

- General: higher performance
  → higher end user acceptance for software [2]
- Special case: Framework
  → Overhead induced on every build software [11]
The WordCounter Application

Introduction » Motivation

Figure: Test configuration: WordCounter.

- 13% higher execution time of v2.0 compared to v1.1.2
The Goals of this Thesis

Introduction ▸ Goals

1. Explanation of Examples for Approaches to Detect Bottlenecks
   ▶ Overview and classification of existing approaches
   ▶ Discussion of advantages and drawbacks

2. Apply Bottleneck Detection Approaches on the WordCounter

3. Solve the bottleneck responsible for the higher execution time of TeeTime 2.0 WordCounter

4. Evaluate the solution
Outline

1. Introduction

2. Foundation
   The Pipe & Filter Architectural Style
   The P&F Framework TeeTime
   Bottleneck Detection Approaches

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
Advantages: Easy ...
- ... reordering [10]
- ... reuse [10]
- ... replacement [10]
- Good support for real concurrency [10]
Advanced Pipelines

Foundation ▶ The Pipe & Filter Architectural Style

Figure: A more complex example pipeline. [1]

Figure: A hierarchical example pipeline. [1]
1. Introduction

2. Foundation
   The Pipe & Filter Architectural Style
   The P&F Framework TeeTime
   Bottleneck Detection Approaches

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
Figure: An excerpt of the TeeTime framework architecture. [12]

- High throughput framework
- Modeling and execution of P&F applications
Figure: Plain test configuration: WordCounter.
Figure: Extended test configuration: WordCounter.
Figure: Extended test configuration: WordCounter.
Connection of arbitrary stages

Inter thread pipes:
  - Function as buffer
  - Synchronize threads

Intra thread pipes:
  - Stores only single element (pointer)
  - Activates successor stage
Outline

1. Introduction

2. Foundation
   - The Pipe & Filter Architectural Style
   - The P&F Framework TeeTime
   - Bottleneck Detection Approaches

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
The Term Bottleneck

Foundation ▶ Bottleneck Detection Approaches

▶ No uniform definition
▶ Often defined by the Approach itself:

Approach dependent definitions

▶ If a stage has the *smallest isolation production rate*, this machine is the bottleneck. [4]
▶ The stage right after the fullest buffer (pipe) is the bottleneck [4, 5]
▶ Li et al. 2007 define it more generally:

Approach independent definition

If a stage has the maximum ratio of overall system performance increment to its own standalone performance increment, then this stage is the bottleneck. [6]
Bottleneck Detection Approaches

Static approach [7][8]
- Changing input data
  → Often used for validation/verification
- Static system or code analysis

Scenario-based approach [6]
- Specific configuration → scenario
- Simulate different scenarios → configuration of best performing one is called bottleneck

Coefficient-based approach [7][8]
- Based on simulation output data → coefficients
- Examples for coefficients:
  - Percentage of active time
  - Average length of active time
  - Longest waiting time of items
1. Introduction

2. Foundation

3. Examples for Bottleneck Detection Approaches
   - Scenarios
   - Coefficients

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
Scenarios for the WordCounter Application

Examples for Bottleneck Detection Approaches ▶ Scenarios

- Different number/size of files
  ➔ specifically for I/O
- Different number of threads

Figure: Execution time for different number of WordCounter stages.
1. Introduction

2. Foundation

3. Examples for Bottleneck Detection Approaches
   Scenarios
   Coefficients

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
Examples for Bottleneck Detection Approaches

- Coefficients

- **Longest waiting time of items**
  - Inaccurate [8]
  - Strong correlation to *blocking* of stages
  - No blocking → system in balance

- **Number of blockings or blocking time**
  - Blocking caused by sending to full pipe
  - Blocking caused by pulling from empty pipe
Percentage of active time
  ▶ Easy to implement [8]
  ▶ But similar percentages for different behavior [8]
  ▶ High effort to gain confidence [8]

Average active time
  ▶ Solves the above mentioned draw backs [8]
Percentage of active time
  - Easy to implement [8]
  - But similar percentages for different behavior [8]
  - High effort to gain confidence [8]

Average active time
  - Solves the above mentioned draw backs [8]

Both are quotients of cumulative active time
Outline

An Extended and Combined Approach

1. Introduction

2. Foundation

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
Store data as list of pairs (time stamp, state) for each stage
Combined Approach

An Extended and Combined Approach

- Store data as list of pairs (time stamp, state) for each stage

- Possibilities:
  - Compute times:
    - **Execution**: Subtract first time stamp from last time stamp
    - **Cumulative**: Add up all single durations of a specific state
    - **Average**: Divide cumulative time by the count of the specific state
    - **Percental**: Divide cumulative time by execution time
  - **Blocked time correlates to fullness of pipes**
  - **Number of times a stage changed to a specific state**
Extended Approach: New State Introduced

An Extended and Combined Approach

Figure: Locations for a state change.
Extended Approach: New State Introduced

An Extended and Combined Approach

Figure: Locations for a state change.

- Stage can be active, blocked or active waiting
Figure: Locations a state change takes place.

- Only store state *change*
- Store active waiting as cumulative duration
Applying our Approach on TeeTime

1. Introduction

2. Foundation

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
Scenario & Experimental Setup

Applying our Approach on TeeTime

- WordCounter for both version of TeeTime
- One test file: 190,692,500 words
- 3 JVM runs, 3 warm ups, 5 measurement runs
- Measurement by StateLogger
The Results

Applying our Approach on TeeTime

Figure: Cumulative active, active waiting and blocked time of the WordCounter. Version 1.1.2 left, version 2.0 right.
Differences in the Data

Applying our Approach on TeeTime

Figure: Difference of the above presented data.

- High blocking time of Merger → Bottleneck lies earlier
- High(er) blocking time of Distributer → Bottleneck lies later
Differences in the Data

Applying our Approach on TeeTime

Figure: Difference of the above presented data.

- **MappingCounter** has highest difference in *active time* → very likely the bottleneck
1. Introduction

2. Foundation

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
   The Culpable Code and a Solution
   Evaluating the Solution
Differences in the Code

A Solution for the TeeTime Bottleneck

The Culpable Code and a Solution

Figure: Differences in the MappingCounter.

- Statical Analysis (Diff) → identical
- Only invoked method: `counter.increment(element)`; → `CountingMap` requires closer look
Differences in the Code

A Solution for the TeeTime Bottleneck ▶ The Culpable Code and a Solution

Figure: Differences in the CountingMap.

▶ TeeTime 2.0 uses a newer implementation of ObjectIntMap
Back to the roots!

→ Use the older data structure
1. Introduction

2. Foundation

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. **A Solution for the TeeTime Bottleneck**
   The Culpable Code and a Solution
   Evaluating the Solution
Scenarios & Experimental Setup

A Solution for the TeeTime Bottleneck ▸ Evaluating the Solution

- WordCounter for both version of TeeTime
  - version 2.0 now also uses old data structure
- One test file: 190,692,500 words
- 3 JVM runs, 3 warm ups, 5 measurement runs
- Measurement by StateLogger
Differences in the Data

A Solution for the TeeTime Bottleneck ↓ Evaluating the Solution

Figure: Difference of the execution time with v2.0 using old data structure.

- Total execution time overhead barely existent
- Other Differences → may be caused by disparities in data collection due to differences in both versions
1. Introduction

2. Foundation

3. Examples for Bottleneck Detection Approaches

4. An Extended and Combined Approach

5. Applying our Approach on TeeTime

6. A Solution for the TeeTime Bottleneck
Many approaches can be combined → mainly a matter of representation

Active waiting gives a better clue of systems behavior

StateLogger is integrated in TeeTime
Bottleneck has been found → hopefully future ones too
Futur Work

- Some anomalies not yet studied
  - TeeTime 2.0 faster then 1.1.2 on many threads
  - Stage behavior differences: noise or real?

- Extension of the StateLogger for TeeTime
  - More representations.
  - Computation of representation could be organized as P&F architecture

- Optimization of the StateLogger
  - Should be choose-able
  - Optimize runtime overhead of the StateLogger itself


Evaluating Approaches to Detect Bottlenecks in the Pipe & Filter Framework TeeTime

Adrian Pegler

March 18, 2016

se...
**Experimental Setup**

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>Intel Core i7-4710 HQ</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>x86-64</td>
</tr>
<tr>
<td><strong>Clock/Core</strong></td>
<td>2.5GHz</td>
</tr>
<tr>
<td><strong>Cores (# of threads)</strong></td>
<td>4 (8)</td>
</tr>
<tr>
<td><strong>RAM</strong></td>
<td>16 GB</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>Windows 10</td>
</tr>
<tr>
<td><strong>JDK</strong></td>
<td>Oracle JDK 1.8.0_60</td>
</tr>
<tr>
<td><strong>VM runs</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>warm ups</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>measurement runs</strong></td>
<td>5</td>
</tr>
</tbody>
</table>

**Table:** Setup used for the evaluation
TeeTime Stages

Figure: Method calls of a consumer stage.
Figure: Method calls between two adjacent Threads.
Figure: Method calls within one single thread.