3C03 Concurrency: Concurrent Architectures: Filter Pipelines

Mark Handley

Outline

- Motivation
- Concurrent Prime Sieve of Eratosthenes
- Modelling Prime Sieve in FSP
- Buffer Tolerance
- Abstraction from Filter Tasks
- Architectural Property Analysis
- Java Prime Sieve Implementation
Software architectures identify software components and their interaction.

- In the context of this course, architectures are process structures, together with the way processes interact.

The aim is to ignore many of the details concerned with specific applications:

- Study structures that can be used in many different situations and applications.

This is the first of three lectures each identifying a particular architectural style.

- Architectural styles are recurring patterns of components and connectors.

We will discuss:

- Filter pipelines
- Supervisor/workers
- Announcer/listener

Each of these commonly occur in concurrent and distributed systems.
Filter Pipelines

- Filters receive one or more input value streams and transform them into one or more output value streams.
  - We only consider filters with one input stream and one output stream.

- Filters are connected by pipelines
  - Redirect output of one filter to input of next
  - May buffer values to de-couple processes from each other.

Example (Unix):
```bash
ypcat passwd | grep 2000_bsc | sort | less
```

Example: Prime Sieve

Goal: compute primes between 2 and N

- Classic algorithm by Eratosthenes known as the Prime Sieve:
  ```c
  for (i=2; i<=N; i++) sieve[i]=i;
  for (i=2; i<=N; i++) {
    if (sieve[i] != 0) print(i);
    for (j=i; j<=N; j++)
      if (sieve[j]%i == 0) sieve[j]=0;
  }
  ```
Prime Sieve

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73

Prime Sieve FSP Model

- General idea:
  - Generate a stream of numbers from 2 to N.
  - Create one filter for each prime number between 2 and N that filters all the numbers that are multiples and only outputs the others.
  - Interconnect Filters by Pipes.

- Leads to Filter Pipeline:
Prime Sieve in FSP

const MAX=5
range NUM=2..MAX
set S={NUM, eos}

Pipe process buffers elements from set S:
PIPE=(put[x:S]->get[x]->PIPE).

GEN process outputs numbers from 2 to MAX followed by the signal eos:
GEN=GEN[2],
GEN[x:NUM]=(out.put[x]->
if x<MAX then
  GEN[x+1]
else
  (out.put.eos->end->GEN)
).

/*initialize from the first input from prev stage*/
  |in.get.eos -> ENDFILTER),

/*filter all inputs that are multiples of p*/
FILTER[p:NUM]=(in.get[x:NUM]->
if x%p!=0 then
  (out.put[x]->FILTER[p])
else
  FILTER[p]
  |in.get.eos->ENDFILTER
),

/*terminate filter on eos*/
ENDFILTER=(out.put.eos -> end -> FILTER).
Prime Sieve in FSP

Glue everything together:

\[
\text{PRIMES}(N=4) = \\
\text{(gen:GEN} \\
\text{pipe}[0..N-1]:PIPE} \\
\text{filter}[0..N-1]:FILTER) \\
\{/pipe[0]/gen.out,} \\
\text{pipe[i:0..N-1]/filter[i].in,} \\
\text{pipe[i:1..N-1]/filter[i-1].out,} \\
\text{end/\{filter[0..N-1].end,gen.end\}} \\
\text{@\{filter[0..N-1].prime,end\}}.
\]

Buffering

- The Prime Sieve Model so far has just one buffer slot.
  - Does it behave the same with no buffering?
  - Does it behave the same with more buffering?
  - Explosion in state space occurs if we attempt to model bigger buffer space in pipes.

- Why use buffering?
  - Performance
    - avoid context switches
    - run filters in parallel
  - Network
    - can’t avoid buffering
Unbuffered pipeline

Remove pipes, and directly plumb one filter into the next:

|| PRIMESUNBUF\(N=4\)
= (gen:GEN
  || filter[0..N-1]:FILTER)
/{ pipe[0]/gen.out.put,
  pipe[i:0..N-1]/filter[i].in.get,
  pipe[i:1..N-1]/filter[i-1].out.put,
  end/{filter[0..N-1].end,gen.end}
 )@{filter[0..N-1].prime,end}.

Abstraction from Application Details

- From an architectural point of view it is not important that integers are passed as buffer elements
- We can abstract away this application detail.
  - Just model that some number is passed.
  - Just model that some numbers are prime.
General Filter Pipeline

Abstract out the details of what is passed down the pipe, and what is actually prime:

\[
|\text{AGEN} = \text{GEN}/(\text{out}.\text{put}/\text{out}.\text{put}[\text{NUM}]). \\
|\text{AFILTER} = \text{FILTER}/(\text{out}.\text{put}/\text{out}.\text{put}[\text{NUM}], \\
\quad \text{in}.\text{get}/\text{in}.\text{get}[\text{NUM}], \\
\quad \text{prime}/\text{prime}[\text{NUM}]). \\
|\text{APIPE} = \text{PIPE}/(\text{put}/\text{put}[\text{NUM}], \text{get}/\text{get}[\text{NUM}]).
\]

As before, but using APIPE, AGEN and AFILTER:

\[
|\text{PRIMES}(N=4) = (\text{gen}:\text{AGEN}||\text{pipe}[0..N-1]:\text{APIPE}|| \\
\quad \text{filter}[0..N-1]:\text{AFILTER} \\
\quad /\text{pipe}[0]/\text{gen}.\text{out}, \\
\quad \text{pipe}[i:0..N-1]/\text{filter}[i].\text{in}, \\
\quad \text{pipe}[i:1..N-1]/\text{filter}[i-1].\text{out}, \\
\quad \text{end}/\{\text{filter}[0..N-1].\text{end}, \text{gen}.\text{end}\} \\
\).
\]

Buffered Pipelines

Multi-stage pipe defined using recursive definition:

\[
|\text{MPIPE}(B=3) = \\
\quad \text{if } B=1 \text{ then APIPE} \\
\quad \text{else } (\text{APIPE}/\{\text{mid}/\text{get}\}||\text{MPIPE}(B-1)/\{\text{mid}/\text{put}\})@\{\text{put}, \text{get}\}.
\]

As before, but using MPIPE:

\[
|\text{PRIMES}(N=4) = (\text{gen}:\text{AGEN}||\text{pipe}[0..N-1]:\text{MPIPE}|| \\
\quad \text{filter}[0..N-1]:\text{AFILTER} \\
\quad /\text{pipe}[0]/\text{gen}.\text{out}, \\
\quad \text{pipe}[i:0..N-1]/\text{filter}[i].\text{in}, \\
\quad \text{pipe}[i:1..N-1]/\text{filter}[i-1].\text{out}, \\
\quad \text{end}/\{\text{filter}[0..N-1].\text{end}, \text{gen}.\text{end}\} \\
\).
\]
Architectural Property Analysis

- Refer to properties for abstract model
  - Concerned with structure and interaction
  - Not with detailed operations

- General properties
  - Absence of deadlocks?
  - Eventual termination?
  - Ordering of results: Do filters produce results in the order in the correct order?

Architectural Properties in FSP

- Absence of deadlocks: As usual
- Termination of the system:
  progress END = {end}
- Production of results in proper order:
  property
  PRIMEP(N=4)=PRIMEP[0],
  PRIMEP[i:0..N]=
    (when(i<N) filter[i].prime->PRIMEP[i+1]
     |end->PRIMEP).
  ||ORDER_CHECK=(PRIMES||PRIMEP).
Java Prime Sieve Implementation

Summary

- Concurrent Software Architectures?
- Filter Pipelines
- Modelling Filters & Pipelines in FSP
- Impact of Buffering
- Abstraction from Filter Tasks
- Architectural Property Analysis
- Java Prime Sieve Implementation
- Buffering
- Next: Supervisor-Worker Architectures