Fuzzy-Based Scheduling Algorithm

Advisor: Professor Ming-Hua Wang

Authors: Ng, Qi-Ming; Yi, John; Zeng, Edward

2010-June

Table of Content

List of Figures

1 Introduction

Scheduling is the allocation of resources to separate tasks to ensure completion for all tasks. Currently, there are many established algorithms for process scheduling. However, there is always a demand for greater efficiency for scheduling to maximize processing resources. Some criteria for determining the best algorithm include turnaround time and waiting time, amongst others. Not only is there a demand for meeting such criteria, it is also helpful to offload decision-making to the scheduler, rather than creating hard rules that don't always give the results that are desirable. Using fuzzy logic helps by offering a convenient way to create an output (in our case an effective priority) from input parameters.

The objective of our project is to define a set of rules for scheduling processes using fuzzy logic to minimize waiting time and turnaround time. Because we have gone over several scheduling algorithms already in this Operating Systems class, the scheduling algorithm we have formulated relate directly to what we have already learned, especially considering the goals of process scheduling we are trying to meet. We plan on having a broad scope of general process scheduling rather than focusing on a particular system environment, hence we are not too concerned with certain criteria such as response time or predictability.

We mainly focus our scope of investigation on fuzzy parameter adjustments, running simulations, and comparing the outputs.

2 Theoretical Bases and Literature Review

There is already a significant amount of research in academia for process scheduling using fuzzy logic, though due to time constraints we have been limited to choosing three papers related to our idea. The existing problem we are addressing is dealing with minimizing wait time and turnaround time with scheduling using fuzzy logic. The papers we found sought to address this by proposing their own input parameters for MATLAB's Fuzzy Logic Toolbox.

One algorithm proposed in the papers was based off the High Response Ratio Next algorithm using fuzzy logic. The input parameters used were the service time and waiting time, similarly to the HRRN algorithm. Although the mentioned they were able to achieve better performance than traditional HRRN, their provided example for their results was very limited. They only provided one example of five processes, which we don't think is enough data to test their algorithm. We think it is too similar to the existing HRRN algorithm to provide significant performance improvements.

Another algorithm using fuzzy logic was geared towards multimedia and real-time operating systems. They proposed using the current static priority, the deadline, and slack time of a process to determine the dynamic priority. The higher the three inputs, the

higher the dynamic priority. Because our general scheduling does not necessarily have all the same requirements as a real-time system, we do not believe the deadline and slack time will be relevant to the dynamic priority in our case.

The last algorithm we looked over used fuzzy logic methodology for batch process scheduling. They used a process's literal priority, critical ratio, queue length, and current relative position in the queue as input parameters for the fuzzy inference system. However, the rule for current relative position had processes near completion having lower effective priority, which we think is unnecessary and causes processes to wait longer than necessary.

We will be using different input parameters to help determine our effective priority using a single processor. We will use the following parameters as inputs for our fuzzy inference system:

- Literal priority: higher leads to higher effective priority.
  - Taken from original paper.
- Current relative position: nearer to completion leads to higher effective priority.
  - This is contrast to the paper's suggestion, because we believe that by completing a (now) shorter job will result one less process in the scheduling queue, thus reducing the overhead.
- Service time: higher leads to higher effective priority.
  - We define this as (accumulated waiting time)/(completion time).
  - We believe that this parameter will increase the fairness of the over scheduling scheme.

3 Hypothesis

Based on the papers we have reviewed, we believe that by taking the combination of the all the algorithms and modifying the parameters will lead to more optimal results. We will use an empirical and common sense approach for parameter modifications. We will attempt to find the fine tuned parameters by running simulation on the instruction set architecture (listed in section 5) that will produce better scheduling results.

4 Methodology

Our approach to the simulation is by taking a scheduling parameter file and run it against several sample assembly-based programs. The simulator output will give us the effectiveness of the scheduling parameters. By tuning scheduling parameters, we can then determine which parameters are better by comparing the simulator outputs.

4.1 Input data

One or multiple assembly-language program shall be taken as input(s) to our program, translated into machine language, and stored into a statically allocated memory which will simulate as instructions stored in memory.

The proceeding section will describe an instruction-set architecture that is supported by our processor simulation, and for producing legal assembly-language program as input data.

4.1.1 Instruction-Set Architecture

Our five-stage processor pipeline will support a very simple instruction-set architecture, but it is general enough to solve complex problems. The instruction set and instruction format are as the following:

There are 3 instruction formats (bit 0 is the least-significant bit). Bits 31-25 are unused for all instructions, and should always be 0.

```
R-type instructions (add, nand):
    bits 24-22: opcode
    bits 21-19: reg A
    bits 18-16: reg B
    bits 15-3:  unused (should all be 0)
    bits 2-0:   destReg

I-type instructions (lw, sw, beq):
    bits 24-22: opcode
    bits 21-19: reg A
    bits 18-16: reg B
    bits 15-0:  offsetField (an 16-bit, 2's complement number with a
                             range of -32768 to 32767)

O-type instructions (halt, noop):
    bits 24-22: opcode
    bits 21-0:  unused (should all be 0)
```

Assembly language name for instruction	Opcode in binary (bits 24, 23, 22)	Action
add (R-type format)	000	add contents of regA with contents of regB, store results in destReg
nand (R-type format)	001	nand contents of regA with contents of regB, store results in destReg
lw (I-type format)	010	load regB from memory. Memory address is formed by adding offsetField with the contents of regA
sw (I-type format)	011	store regB into memory. Memory address is formed by adding offsetField with the contents of regA
beq (I-type format)	100	if the contents of regA and regB are the same, then branch to the address PC+1+offsetField, where PC is the address of the beq instruction

jalr (J-type format)	101	First store PC+1 into regB, where PC is the address of the jalr instruction. Then branch to the address now contained in regA. Note that if regA is the same as regB, the processor will first store PC+1 into that register, then end up branching to PC+1
halt (O-type format)	110	increment the PC (as with all instructions), then halt the machine (let the simulator notice that the machine halted)
noop (O-type format)	111	do nothing

Table: Description of Machine Instructions

4.1.2 Assembly-Language Program Format and Example

The format for a line of assembly code is (<white> means a series of tabs and/or spaces):

```
label<white>instruction<white>field0<white>field1<white>field2<white>comments
```

Here is an assembly-language program example that counts down from 5, stopping when it hits 0:

```
        lw      0       1       five    load reg1 with 5 (symbolic address)
        lw      1       2       3       load reg2 with -1 (numeric address)
start   add     1       2       1       decrement reg1
        beq     0       1       2       goto end of program when reg1==0
        beq     0       0       start   go back to the beginning of the loop
        noop
done    halt                            end of program
five    .fill   5
neg1    .fill   -1
stAddr  .fill   start                   will contain the address of start (2)
```

And here is the corresponding machine language that will be stored in "memory":

```
(address 0): 8454151  (hex 0x810007)
(address 1): 9043971  (hex 0x8a0003)
(address 2): 655361   (hex 0xa0001)
(address 3): 16842754 (hex 0x1010002)
(address 4): 16842749 (hex 0x100fffd)
(address 5): 29360128 (hex 0x1c00000)
(address 6): 25165824 (hex 0x1800000)
(address 7): 5        (hex 0x5)
(address 8): -1       (hex 0xffffffff)
(address 9): 2        (hex 0x2)
```

4.2 Simulator Design

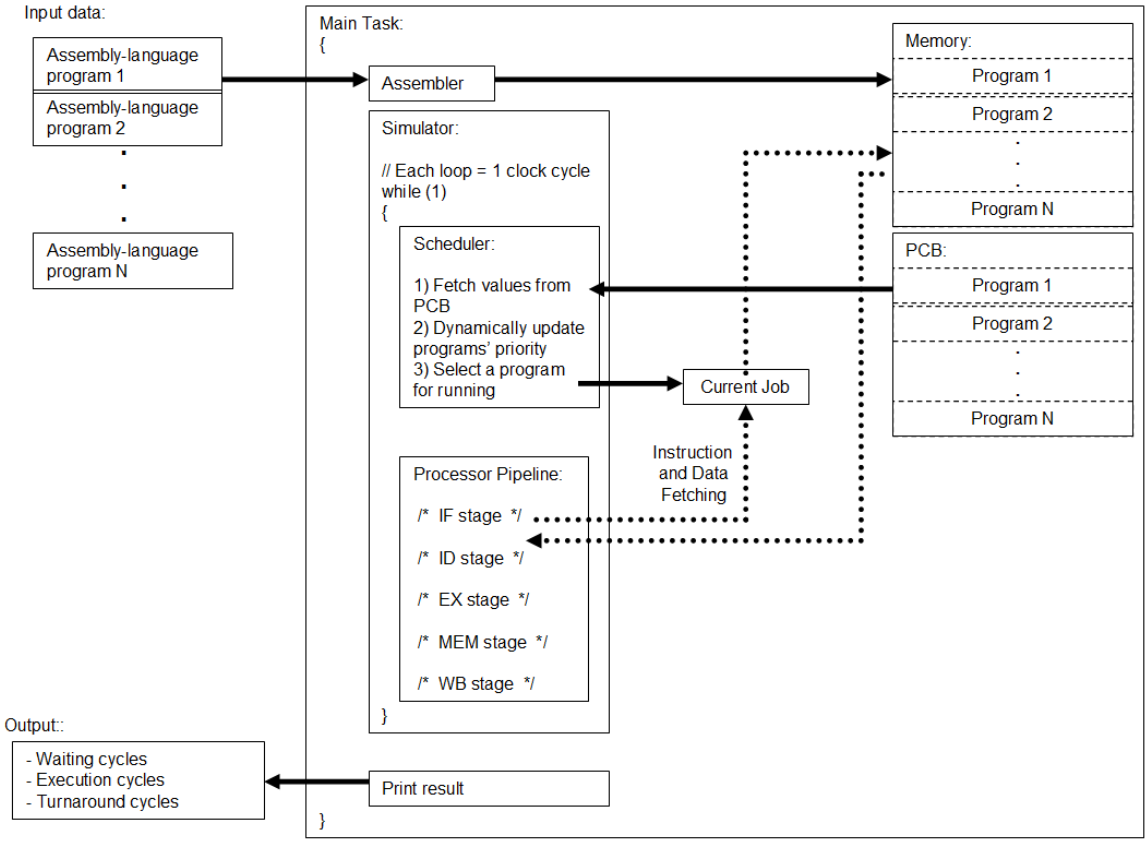

Figure 1 Block Diagram for Simulator Design

Our simulator will consist of:

- An assembler – for assembling assembly-language programs into machine language
- A Fuzzy Scheduler – for dynamically scheduling programs for execution, based on information from the Processor Control Block (PCB)
- A 32-bit Processor – for executing instructions for the current program that has been scheduled by the scheduler

4.2.1 Language Used

- C/C++
- Python

4.2.2 Tools Used

- None

4.2.3 Running the Program

The simulator program takes multiple command-line arguments. Each argument represents the file name where the assembly-language program is stored, in first-come order:

```
> ./project <fcfs|fuzzy|random|robin> program1.as program2.as …
```

The first argument represents the scheduling algorithm to be used, and the rest of the parameters are assembly programs.

- fcfs = First-Come-First-Serve
- fuzzy = Fuzzy-Based
- random = Random
- robin = Round-Robin

The simulator will simulate the program until all assembly-language programs that were passed into the simulator executes a halt.

4.3 Output Generation

For each program that has halted, the simulator will output its execution cycles, waiting cycles and turnaround cycles.

When all programs have halted, the average waiting cycles and average turnaround cycles will be computed and generated as output.

4.4 Testing Against Hypothesis

Average waiting cycles and average turnaround cycles that have been computed will be used to test the performance of our scheduler.

5 Implementation

We made use of the open source Free Fuzzy Logic Library (FFLL) from http://ffll.sourceforge.net. We constructed the fuzzy parameters and fuzzy rule as input to the FFLL. We randomized the literal priority for each executable to make the simulation more realistic. The relative position and service time are dynamically calculated and send to the fuzzy engine to derive the effective priority. The following figure depicts an architectural view of the simulation process.

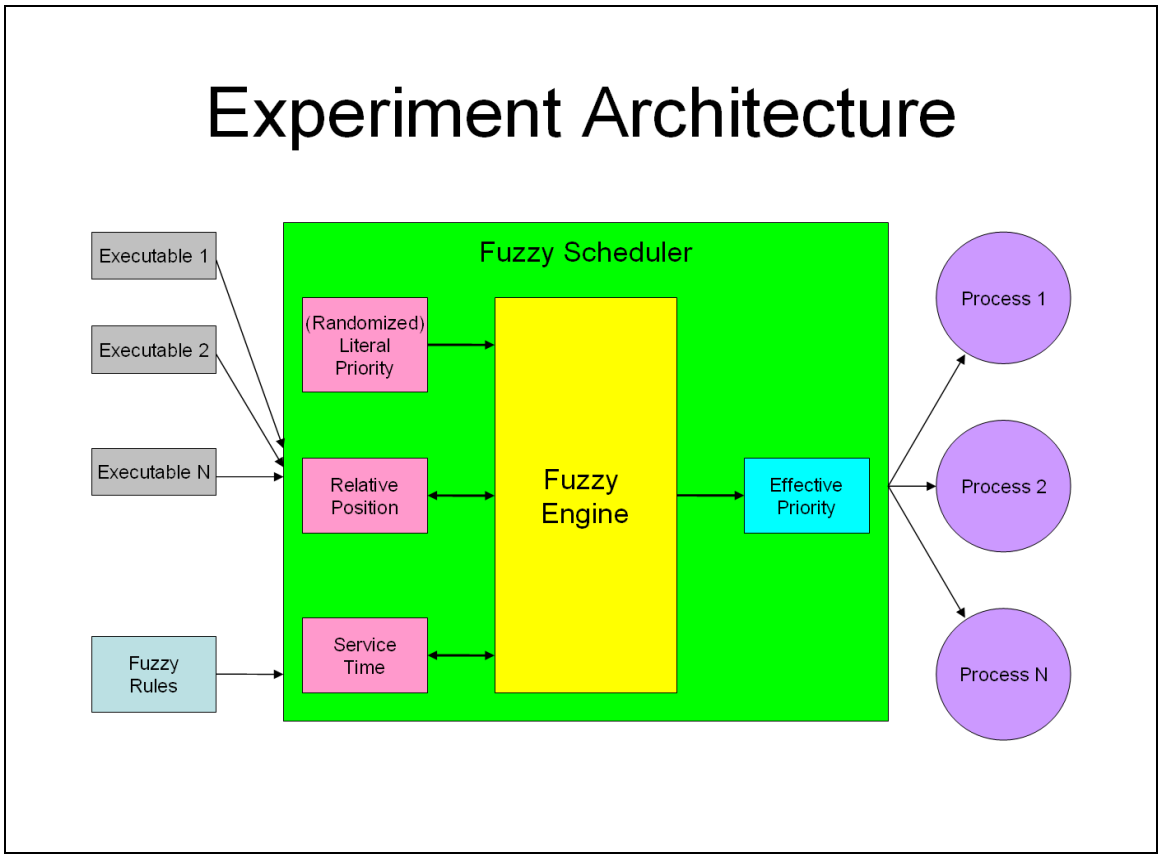

Figure 2 Experiment Architecture

We defined the fuzzy parameters as follows:

```
FUZZIFY Literal_Priority
        TERM Low := (0, 0) (0, 1) (50, 0) ;
        TERM Medium := (14, 0) (50, 1) (83, 0) ;
        TERM High := (50, 0) (100, 1) (100, 0) ;
END_FUZZIFY

FUZZIFY Relative_Position
        TERM Low := (0, 0) (0, 1) (50, 0) ;
        TERM Medium := (14, 0) (50, 1) (83, 0) ;
        TERM High := (50, 0) (100, 1) (100, 0) ;
END_FUZZIFY

FUZZIFY Service_Time
        TERM Low := (0, 0) (0, 1) (50, 0) ;
        TERM Medium := (14, 0) (50, 1) (83, 0) ;
        TERM High := (50, 0) (100, 1) (100, 0) ;
END_FUZZIFY

FUZZIFY Effective_Priority
        TERM Low := 0 ;
        TERM High := 100 ;
END_FUZZIFY
```

We defined the fuzzy rule as follows:

```
IF (Literal_Priority IS High) AND (Relative_Position IS High) AND
```

```
(Service_Time IS High) THEN (Effective_Priority IS High);
IF (Literal_Priority IS High) AND (Relative_Position IS High) AND
(Service_Time IS Medium) THEN (Effective_Priority IS High);
IF (Literal_Priority IS High) AND (Relative_Position IS High) AND
(Service_Time IS Low) THEN (Effective_Priority IS High);
IF (Literal_Priority IS High) AND (Relative_Position IS Medium) AND
(Service_Time IS High) THEN (Effective_Priority IS High);
IF (Literal_Priority IS High) AND (Relative_Position IS Medium) AND
(Service_Time IS Medium) THEN (Effective_Priority IS High);
IF (Literal_Priority IS High) AND (Relative_Position IS Medium) AND
(Service_Time IS Low) THEN (Effective_Priority IS High);
IF (Literal_Priority IS High) AND (Relative_Position IS Low) AND
(Service_Time IS High) THEN (Effective_Priority IS High);
IF (Literal_Priority IS High) AND (Relative_Position IS Low) AND
(Service_Time IS Medium) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS High) AND (Relative_Position IS Low) AND
(Service_Time IS Low) THEN (Effective_Priority IS Low);

IF (Literal_Priority IS Medium) AND (Relative_Position IS High) AND
(Service_Time IS High) THEN (Effective_Priority IS High);
IF (Literal_Priority IS Medium) AND (Relative_Position IS High) AND
(Service_Time IS Medium) THEN (Effective_Priority IS High);
IF (Literal_Priority IS Medium) AND (Relative_Position IS High) AND
(Service_Time IS Low) THEN (Effective_Priority IS High);
IF (Literal_Priority IS Medium) AND (Relative_Position IS Medium)
AND (Service_Time IS High) THEN (Effective_Priority IS High);
IF (Literal_Priority IS Medium) AND (Relative_Position IS Medium)
AND (Service_Time IS Medium) THEN (Effective_Priority IS High);
IF (Literal_Priority IS Medium) AND (Relative_Position IS Medium)
AND (Service_Time IS Low) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Medium) AND (Relative_Position IS Low) AND
(Service_Time IS High) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Medium) AND (Relative_Position IS Low) AND
(Service_Time IS Medium) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Medium) AND (Relative_Position IS Low) AND
(Service_Time IS Low) THEN (Effective_Priority IS Low);

IF (Literal_Priority IS Low) AND (Relative_Position IS High) AND
(Service_Time IS High) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Low) AND (Relative_Position IS High) AND
(Service_Time IS Medium) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Low) AND (Relative_Position IS High) AND
(Service_Time IS Low) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Low) AND (Relative_Position IS Medium) AND
(Service_Time IS High) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Low) AND (Relative_Position IS Medium) AND
(Service_Time IS Medium) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Low) AND (Relative_Position IS Medium) AND
(Service_Time IS Low) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Low) AND (Relative_Position IS Low) AND
(Service_Time IS High) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Low) AND (Relative_Position IS Low) AND
(Service_Time IS Medium) THEN (Effective_Priority IS Low);
IF (Literal_Priority IS Low) AND (Relative_Position IS Low) AND
(Service_Time IS Low) THEN (Effective_Priority IS Low);
```

6 Data Analysis and Discussion

We used the waiting time as the performance factor. Smaller waiting time indicates better performance. We compared the program output of the following four scheduling algorithms:

- First-Come-First-Serve
- Fuzzy-Based
- Random
- Round-Robin

The fuzzy-based and random algorithms were executed multiple times, and the mean waiting time is calculated by the Python script analyze.py. Based on several simulation iterations, we obtained consistent comparison measurements as follows:

- Round Robin
  - Approximately 15% better
- Random
  - Approximately 15% better
- First Come First Serve
  - Approximately 8% worse
  - Due 0 waiting time on 1st, small on 2nd, …
  - This scheduling algorithm is mostly never deployed on real systems due to unfairness

7 Conclusion and Recommendations

Based on our experiment, we concluded that the fuzzy-based scheduling algorithm does indeed perform better than the commonly used algorithms, such as round-robin.

We believe that similar results can be achieved by comparing to other scheduling algorithms such as weighted-round-robin, priority-based, and multilevel queue.

Other improvements can be made by additional fuzzy parameters or fine tuning the fuzzy rules.

Lastly, our focus on this project was on developing a fuzzy-based scheduler. As a result, an overly simplified processor simulation was implemented. Some future work may include:

- Adding context switching time
- Adding wait cycles for reading and writing to memory
- Enhancing processor with caching scheme

8 Bibliography

Mohammad Atique, and Mir Sadique Ali, “A Novel Adaptive Neuro Fuzzy Inference

System Based CPU Scheduler for Multimedia Operating System", Proceeding for the International Joint Conference on Neural Networks, Orlando, Florida, USA, August 2007

Abdurazzag Ali Aburas, and Vladimir Miho, "Fuzzy Logic Based Algorithm for Uniprocessor Scheduling", Proceedings of the International Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia, May 2008

Bo Zhang, "A Fuzzy-Logic-Based Methodology for Batch Process Scheduling", Proceedings of the 2006 Systems and Information Engineering Design Symposium, Los Angeles, California, USA, 2006

Introduction to Computer Organization, http://www.eecs.umich.edu/~mahlke/370f01/

MATLAB 6.5 Fuzzy Logic Toolbox, http://www.MathWorks.com