A Comprehensive Study of Characterizing Program Execution Time

Young-Kyoon Suh

May 3, 2016

Abstract

(Tentative) Measuring execution time is a useful tool in evaluating the performance of a program. But it is challenging to obtain precise, accurate program execution time due to the presence of various system daemons and their unpredictable activities. Such activities significantly contribute to varying program execution time. It will be very useful to predict the concrete performance of a program with different input sizes in a real situation, if a (probabilistic) distribution of the execution times of that program is found, considering the daemon activities. In this work, we discuss several interesting phenomenon observed to characterize program execution times. We find that such a distribution of the execution times cannot be uniquely identified, and it will be more likely to be mixture of two or more models formed by different periodicity of different daemon processes. Finally, we discuss some remaining issues that should be resolved to successfully identify a distribution of program execution times.

1 Experiment Notes

Task Length	Description	Time Length
Regular PUT experiment. Refer to Sections 2, 3, and 4.		
PUT1∼PUT64	Runs of 1000 samples (on `sodb12`).	2013-10-14 ∼ 2013-10-15
PUT128∼PUT2048	Runs of 300 samples (on `sodb12`).	2013-12-12 ∼ 2013-12-21
PUT4096	A run of 300 samples (on `sodb12`).	2014-06-23 ∼ 2014-07-10
PUT8192	Runs of 40/260 samples (on `sodb12`).	2015-04-23 ∼ 2015-04-27 / 2015-10-31 ∼ 2015-11-24
PUT16384	Runs of 40/260 samples (on `sodb12`).	2015-04-23 ∼ 2015-04-23 / 2015-11-25 ∼ 2016-01-14

Table 1: Notes on the regular PUT data used for the histograms

Task Length	Description	Time Length
Regular PUT experiment. Refer to Section 5.		
PUT1/2	Runs of 20k samples on `sodb9`/`sodb10`.	2015-12-15 ∼ 2015-12-15
PUT4/8	Runs of 20k samples on `sodb10`.	2016-01-20 ∼ 2016-01-20
PUT16	Runs of 2k, 4k, 8k, 16k, and 32k samples (on `sodb12`).	2016-01-25 ∼ 2016-02-09
Dual PUT experiment. Refer to Section 6.		
PUT4096	A run of 500 samples on `sodb8`.	2015-11-08 ∼ 2015-12-25
PUT2	A run of 1k samples on `sodb9`.	2015-12-27 ∼ 2015-12-27
PUT4∼PUT32	Runs of 1k samples on `sodb9`.	2016-01-27 ∼ 2016-01-31
PUT64∼PUT2048	Runs of 1k samples on `sodb9`.	2016-02-17 ∼ 2016-04-05
PUT2048	A run of 100 samples on `sodb9`.	2016-04-13 ∼ 2016-04-16
PUT4096	A run of 100 samples on `sodb8`.	2016-04-13 ∼ 2016-04-19

Table 2: Notes on the new PUT experiments

2 Summary of the EMPv4 data

EMPv4: Running PUT with a specific task length under a controlled environment, with i) daemon processes disabled, ii) the NTP daemon process activated, iii) major CPU features (turbo and speedstep) disabled, and iv) an up-to-date Linux version (RHEL 6.0) installed.

	Num. of Samples	Minimum (msec)	Maximum (msec)	Average (msec)	Std. Dev. (msec)
PUT1	1,000	999.0	1,005.0	1,002.4	0.73
PUT2	1,000	1,996.0	2,007.0	2,004.5	1.38
PUT4	1,000	4,004.0	4,012.0	4,008.6	1.64
PUT8	1,000	8,014.0	8,023.0	8,018.1	1.72
PUT16	1,000	16,029.0	16,041.0	16,034.3	1.86
PUT32	1,000	32,064.0	32,084.0	32,068.2	2.05
PUT64	1,000	64,129.0	64,145.0	64,135.0	2.27
PUT128	300	128,244.0	128,260.0	128,251.2	2.32
PUT256	300	256,494.0	256,523.0	256,502.3	3.29
PUT512	300	512,995.0	513,152.0	513,005.1	9.41
PUT1024	300	1,025,997.0	1,026,141.0	1,026,012.4	11.43
PUT2048	300	2,051,981.0	2,052,156.0	2,052,012.0	11.19
PUT4096	300	4,105,451.0	4,105,629.0	4,105,526.0	25.98
PUT8192	40 (last Apr)	8,207,870.0	8,207,967.0	8,207,918.0	21.03
PUT8192	260 (Nov)	8,210,940.0	8,211,196.0	8,211,049.0	36.60
PUT16384	40 (last Apr)	16,415,757.0	16,415,964.0	16,415,810.3	40.43
PUT16384	260 (Nov)	16,422,028	16,422,389	16,422,153.0	52.54

Table 3: PT statistics by EMPv4 (See Table 1.)

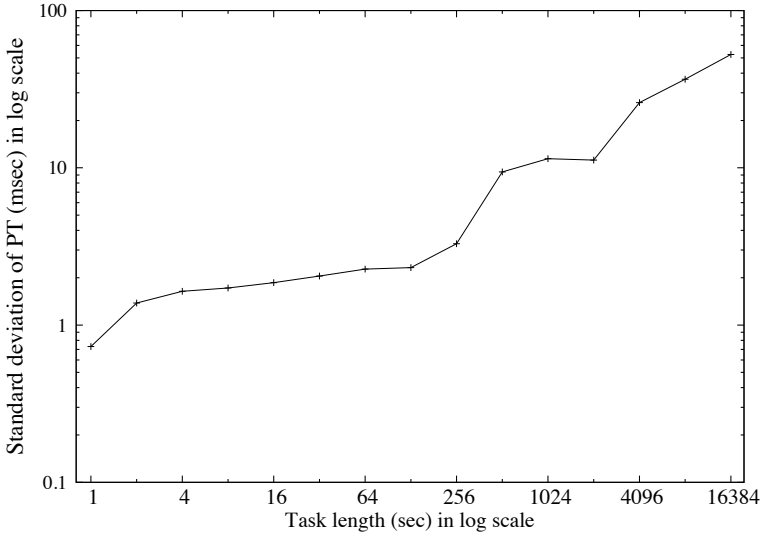
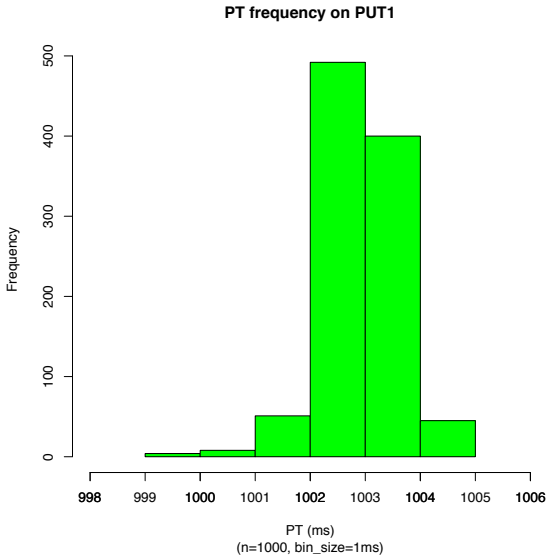


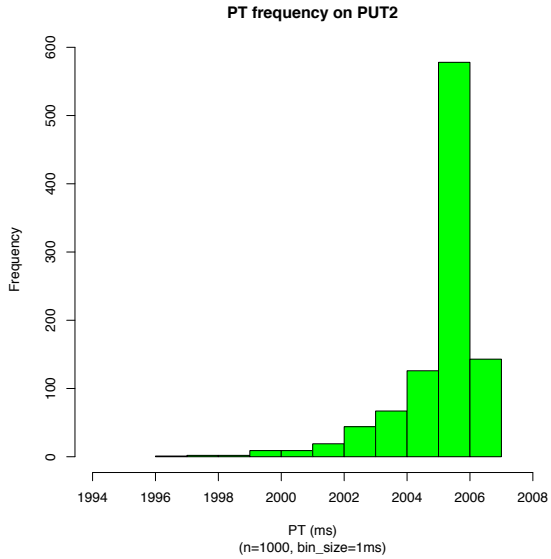
Figure 1: Std. dev. of PT over increasing task length (See Table 1.)

3 Histograms on the EMPv4 Data

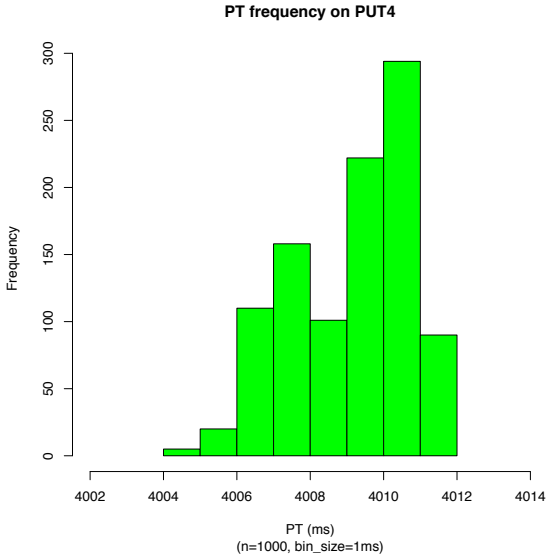
The base data of the following histograms are from Table 1.



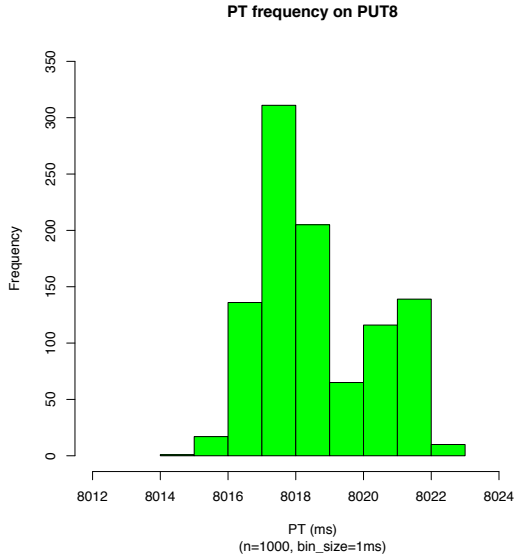
(a) PT frequency on PUT1



(b) PT frequency on PUT2

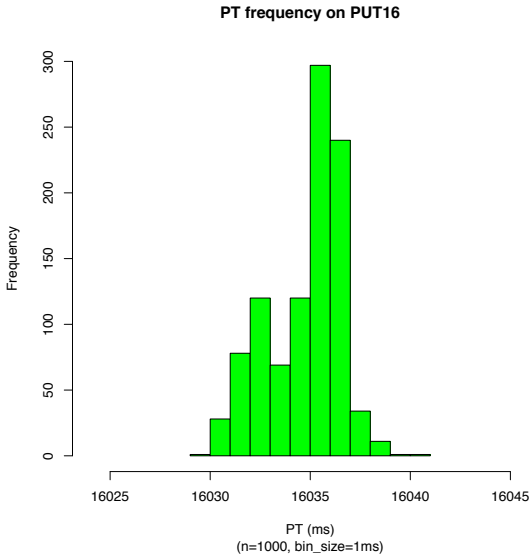


(c) PT frequency on PUT4

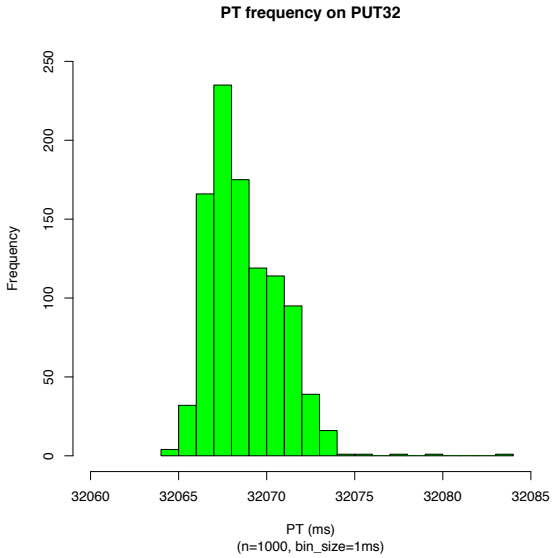


(d) PT frequency on PUT8

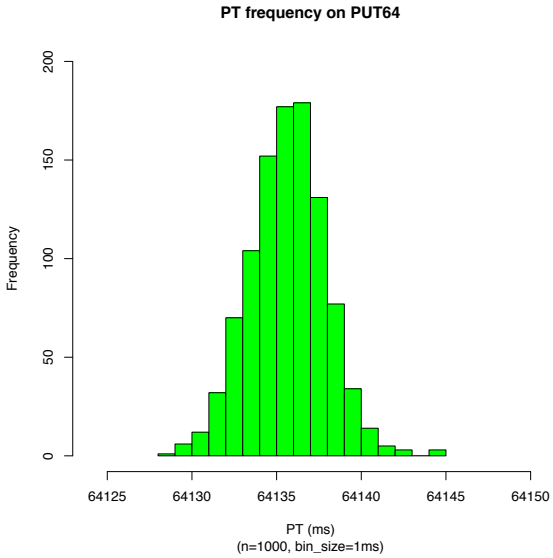
Figure 2: PT Histograms of PUT1 ... PUT8



(a) PT frequency on PUT16

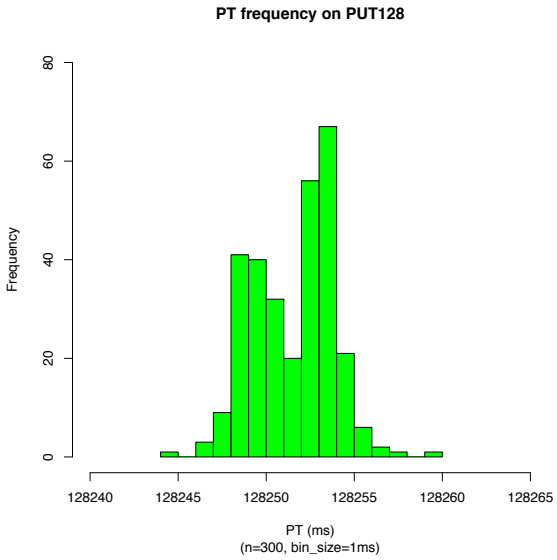


(b) PT frequency on PUT32

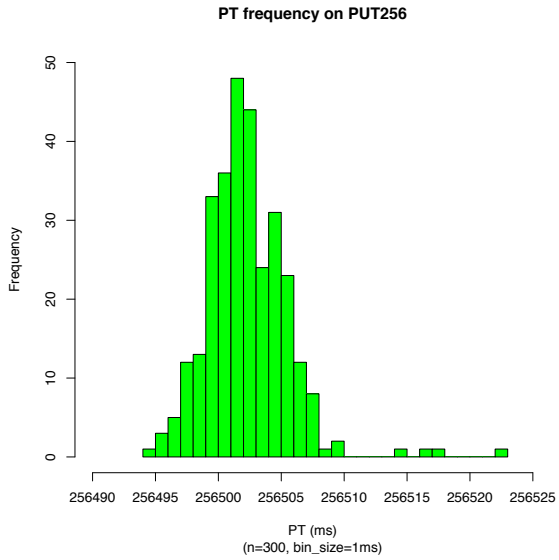


(c) PT frequency on PUT64

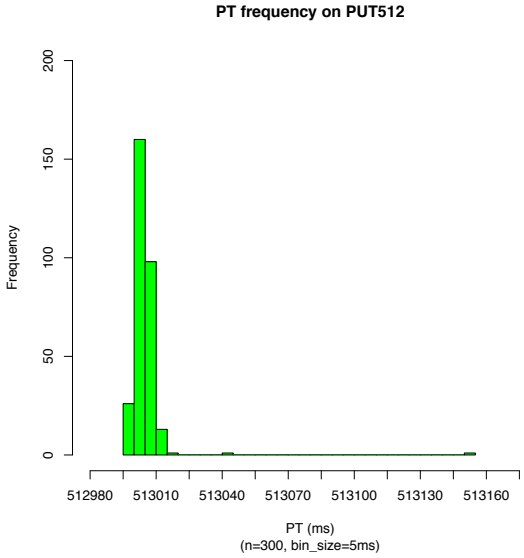
Figure 3: PT Histograms of PUT16 ... PUT64



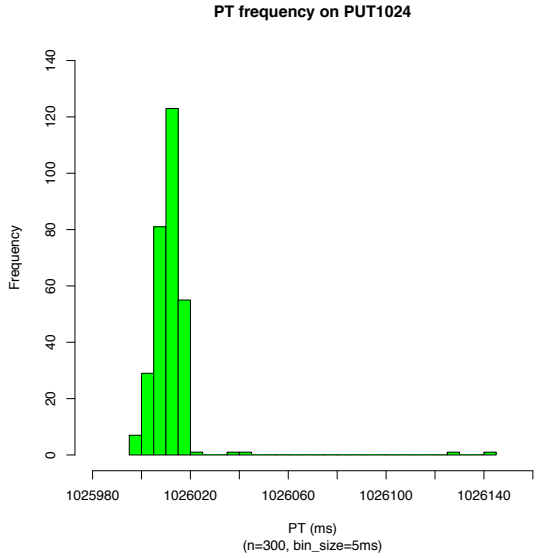
(a) PT frequency on PUT128



(b) PT frequency on PUT256

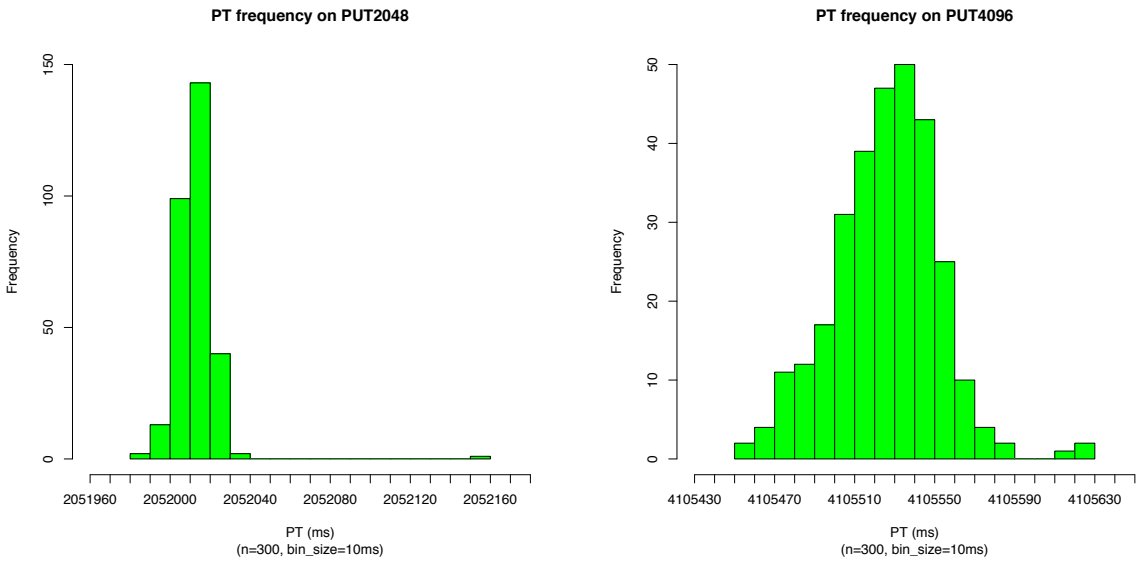


(c) PT frequency on PUT512



(d) PT frequency on PUT1024

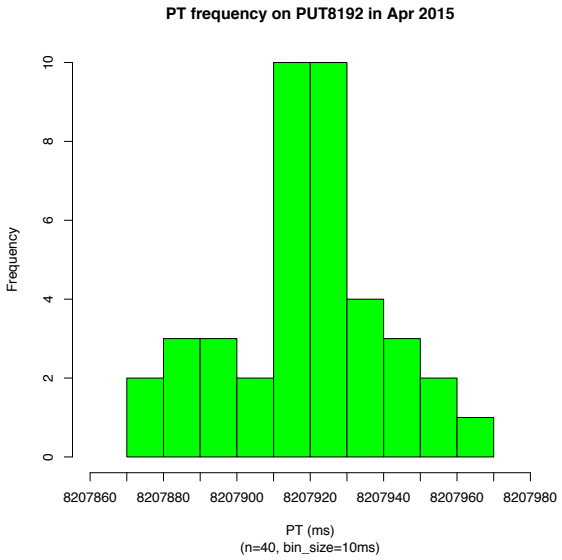
Figure 4: PT Histograms of PUT128 ... PUT1024



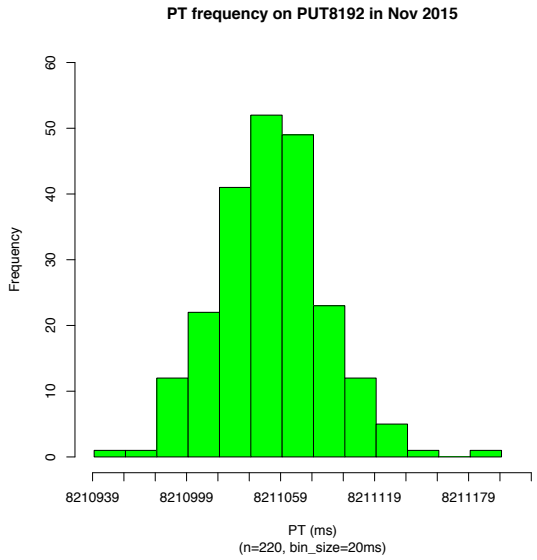
(a) PT frequency on PUT2048

(b) PT frequency on PUT4096

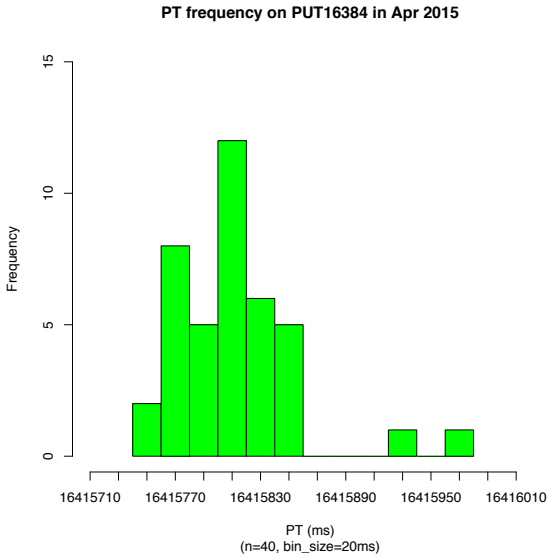
Figure 5: PT Histograms of PUT2048 and PUT4096



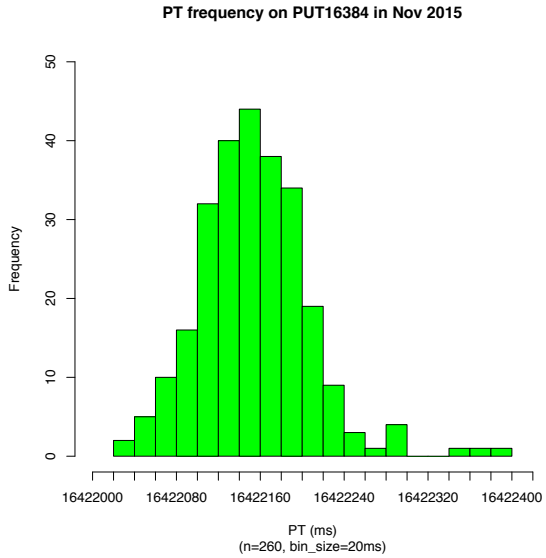
(a) PT frequency on PUT8192 with 40 samples (See Table 1.)



(b) PT frequency on PUT8192 with 260 samples (See Table 1.)



(c) PT frequency on PUT16384 with 40 samples (See Table 1.)

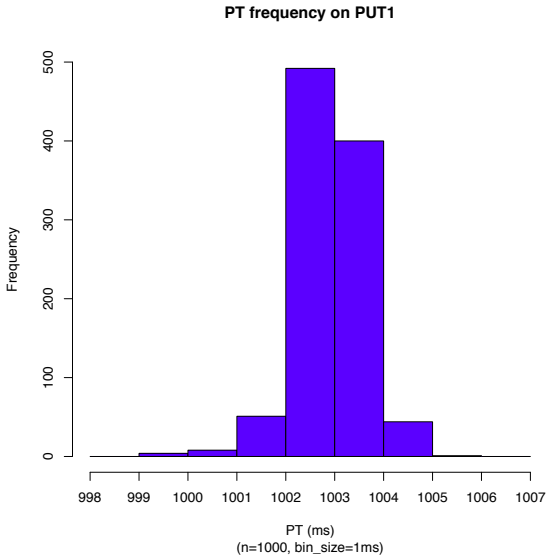


(d) PT frequency on PUT16384 with 260 samples (See Table 1.)

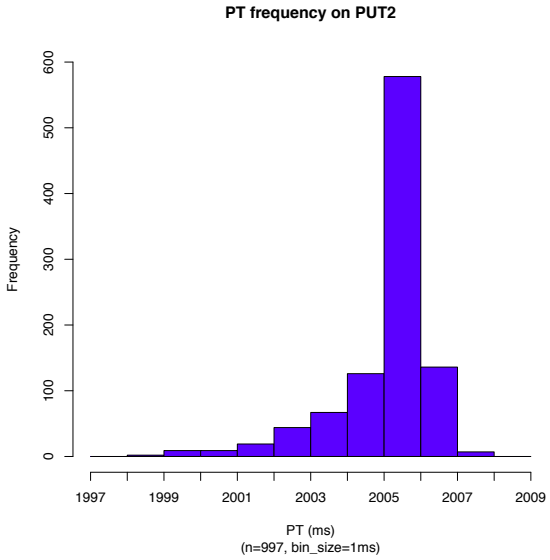
Figure 6: PT Histograms of PUT8192 and PUT16384

4 Histograms on the EMPv5 Data

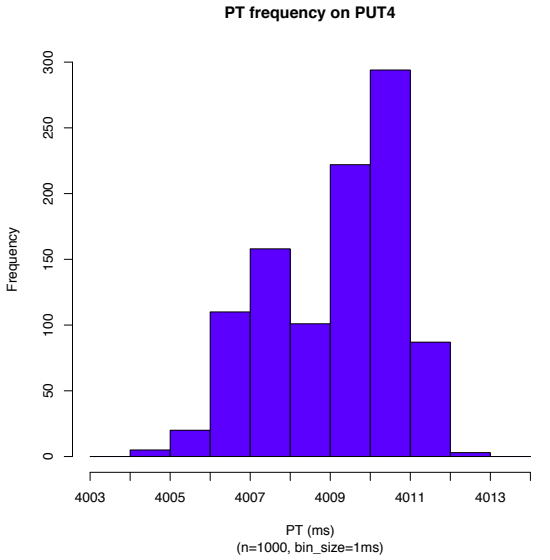
The base data of the following histograms are from Table 1. EMPv5(-relaxed) trims outliers from the data of each PUT by EMPv4. To be more specific, for each run of PUT an outlier is determined as the one above and below the average ± *five[1]* standard deviations computed from the EMPv4 data.



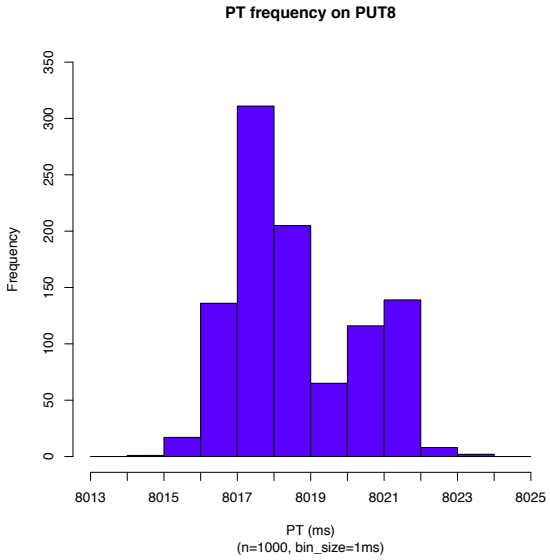
(a) PT frequency on PUT1



(b) PT frequency on PUT2



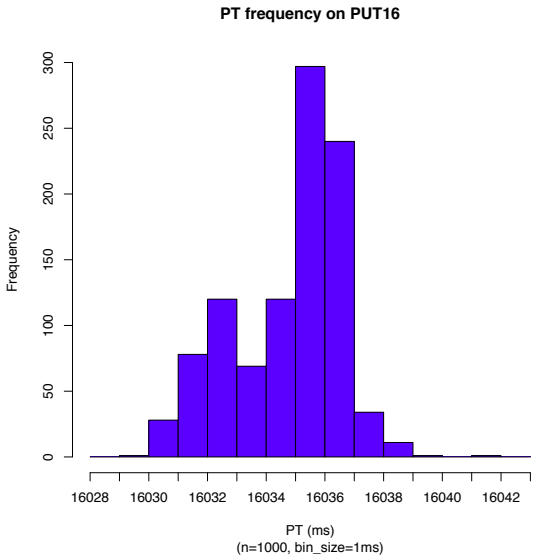
(c) PT frequency on PUT4



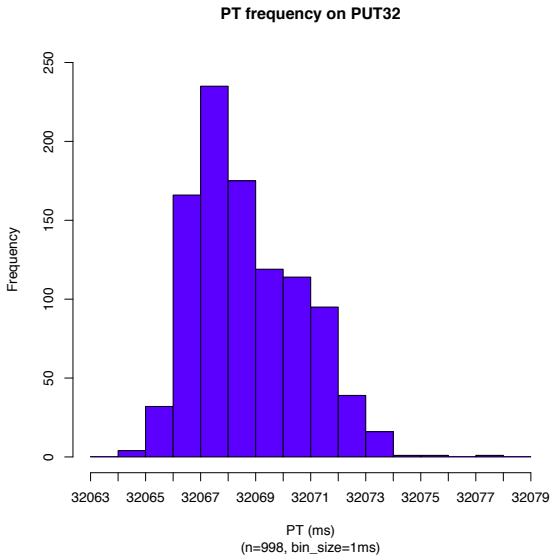
(d) PT frequency on PUT8

Figure 7: PT Histograms of PUT1 ... PUT8

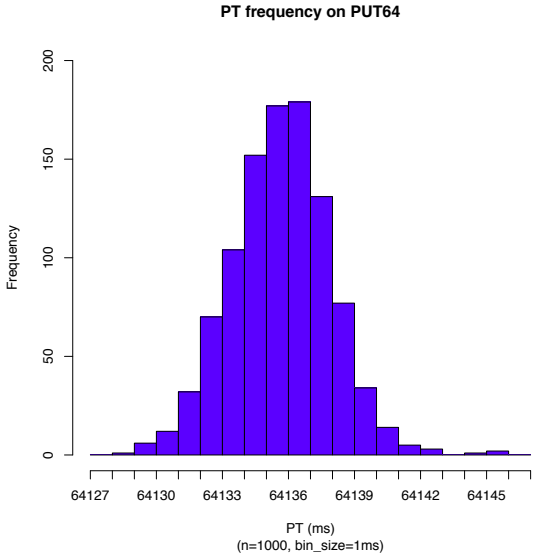
[1]In the stricter version, we use *two*.



(a) PT frequency on PUT16

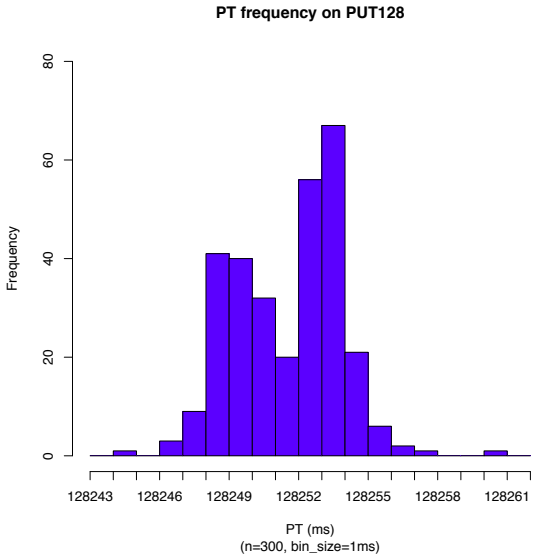


(b) PT frequency on PUT32

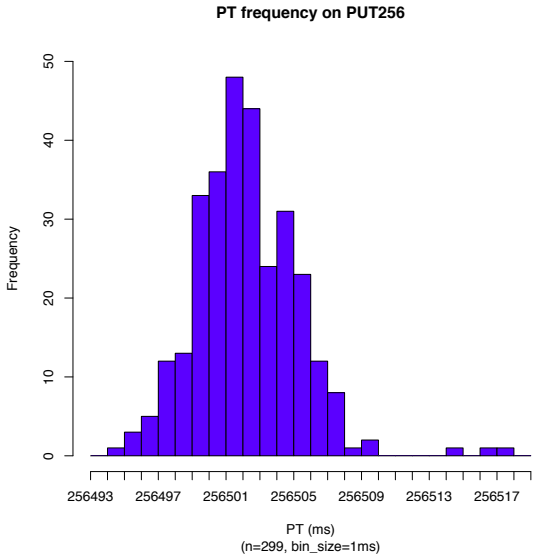


(c) PT frequency on PUT64

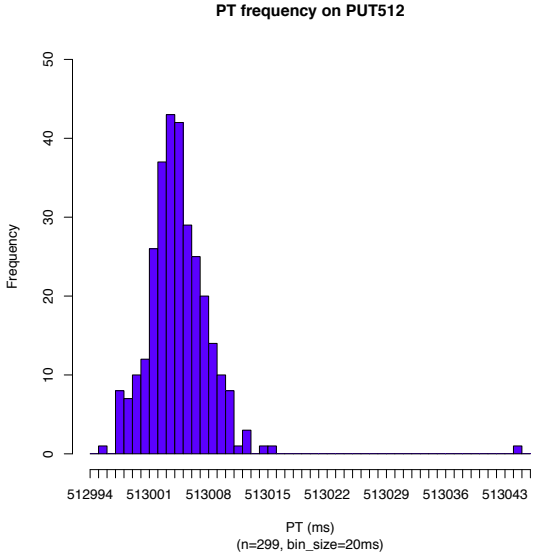
Figure 8: PT Histograms of PUT16 ... PUT64



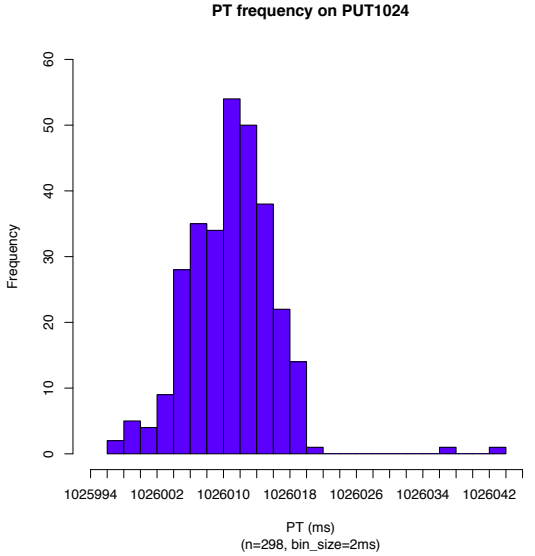
(a) PT frequency on PUT128



(b) PT frequency on PUT256

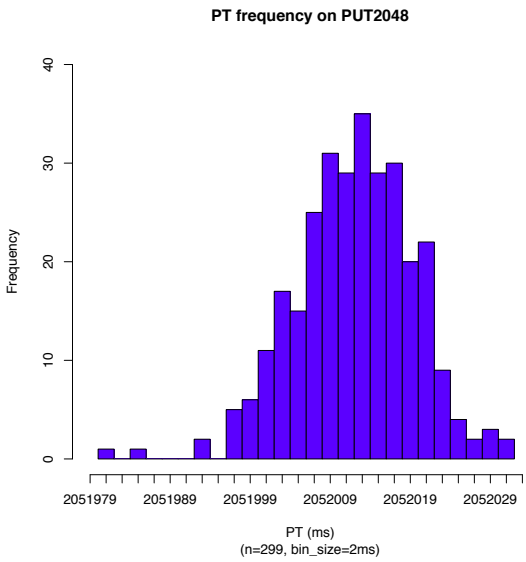


(c) PT frequency on PUT512

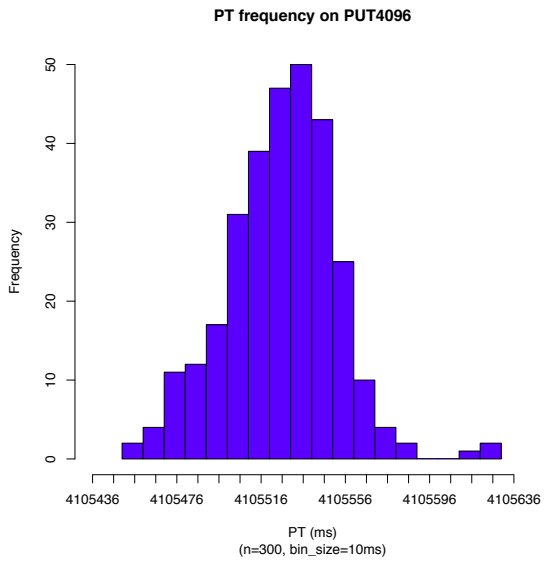


(d) PT frequency on PUT1024

Figure 9: PT Histograms of PUT128 ... PUT1024

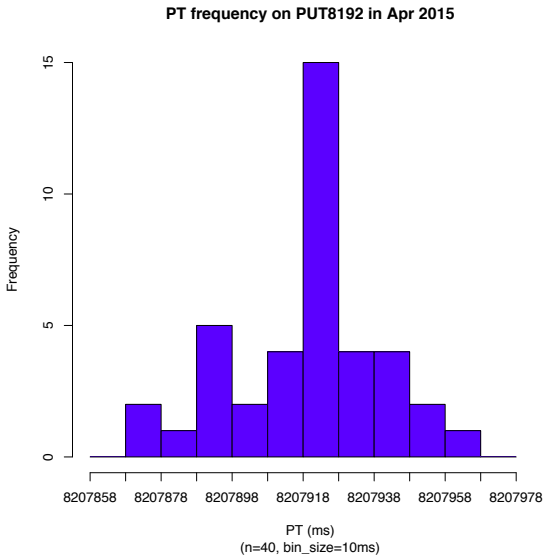


(a) PT frequency on PUT2048

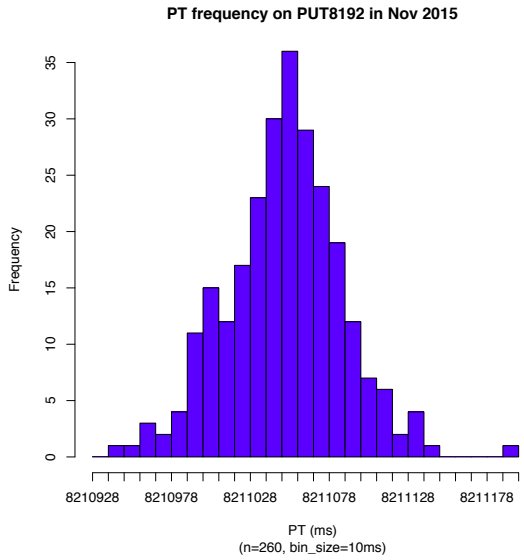


(b) PT frequency on PUT4096

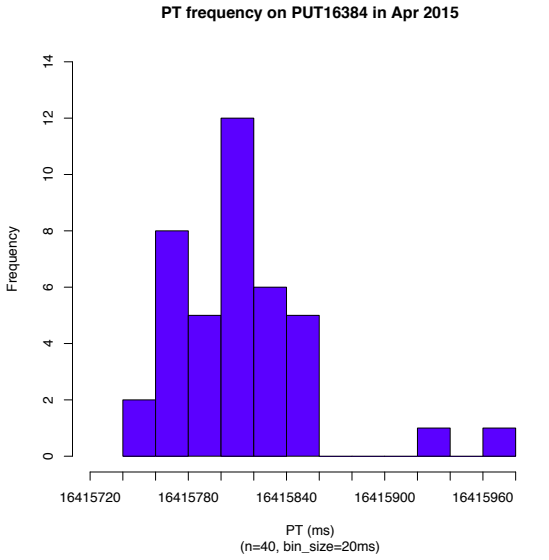
Figure 10: PT Histograms of PUT2048 and PUT4096



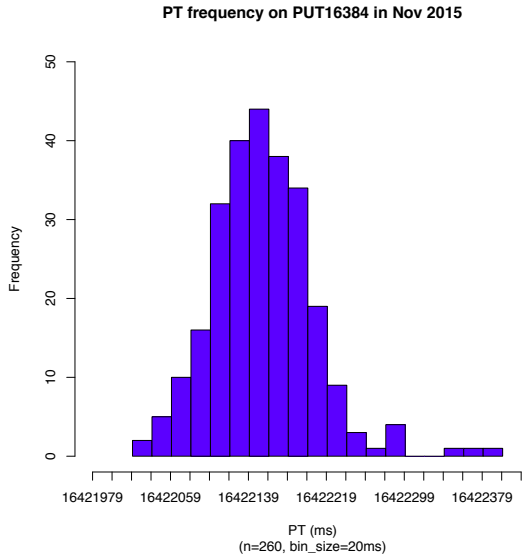
(a) PT frequency on PUT8192 with 40 samples (See Table 1.)



(b) PT frequency on PUT8192 with 260 samples (See Table 1.)



(c) PT frequency on PUT16384 with 40 samples (See Table 1.)



(d) PT frequency on PUT16384 with 260 samples (See Table 1.))

Figure 11: PT Histograms of PUT8192 and PUT16384

5 Sample Size vs. Standard Deviation of PT

The base data of the following histograms are from Table 2.

5.1 PUT1 and PUT2

Table 4 exhibits varying standard deviations over increasing sample size on PUT1 and PUT2. EMPv4 is applied to the table's data.

Num. of Samples	Std. Dev. (msec)	
	PUT1	PUT2
1,000	1.07	1.40
2,000	1.06	1.39
3,000	1.07	1.38
4,000	1.07	1.37
5,000	1.07	1.40
6,000	1.06	1.70
7,000	1.06	1.65
8,000	1.07	1.62
9,000	1.07	1.60
10,000	1.07	1.58
11,000	1.08	1.57
12,000	1.08	1.56
13,000	1.08	1.54
14,000	1.08	1.53
15,000	1.08	1.52
16,000	1.08	1.51
17,000	1.08	1.50
18,000	1.08	1.50
19,000	1.08	1.50
20,000	1.08	1.49

Table 4: Std. Dev. of PUT1 and PUT2 over increasing sample size

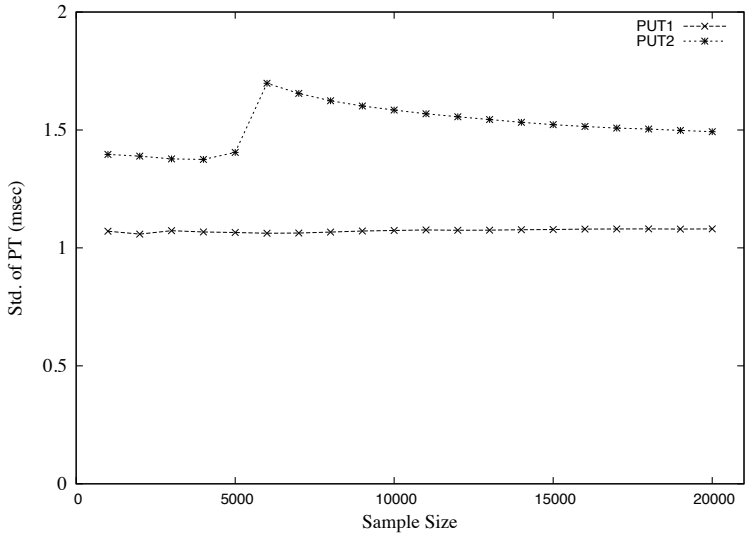


Figure 12: Std. dev. of PT on PUT1 and PUT2 over increasing sample size

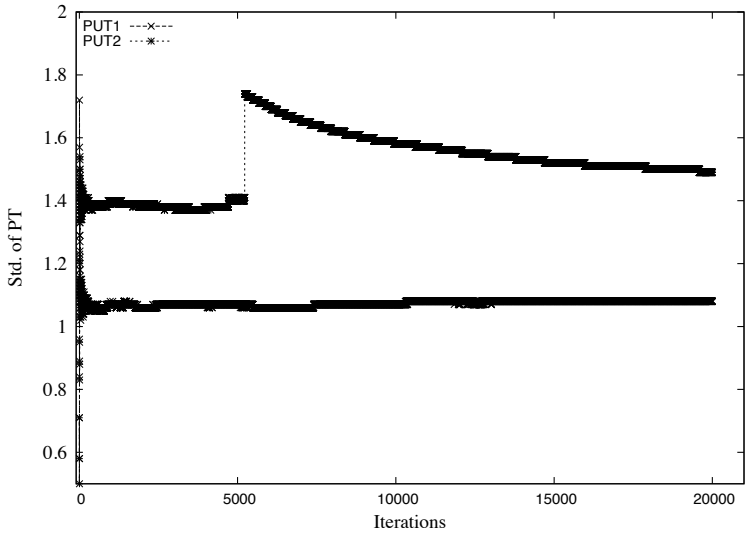
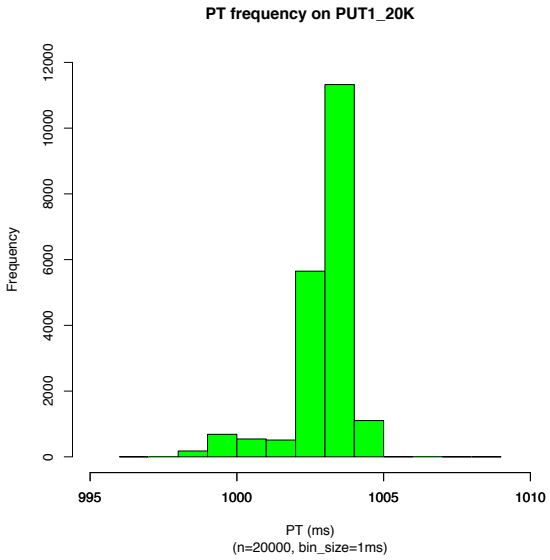


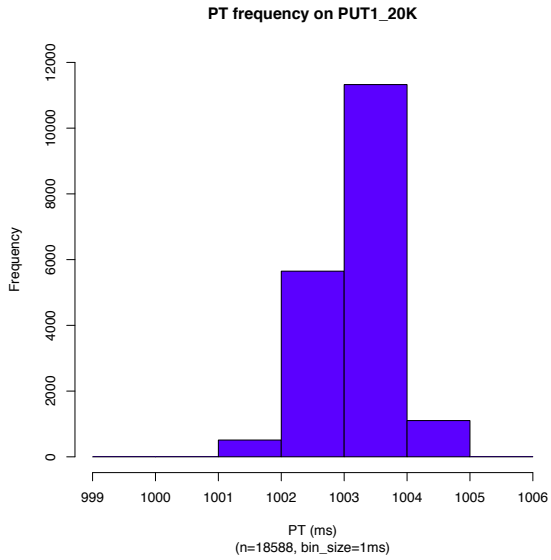
Figure 13: Std. dev. of PT on PUT1 and PUT2 over increasing sample size

PUT2	Program Time
`incr_work`	2078 msecs (at the 5276th iteration)
Daemon Processes	Program Time
`md0_raid1`	1 msec
`proc_monitor`	198 msecs
`rhn_check`	460 msecs
Total	659 msecs

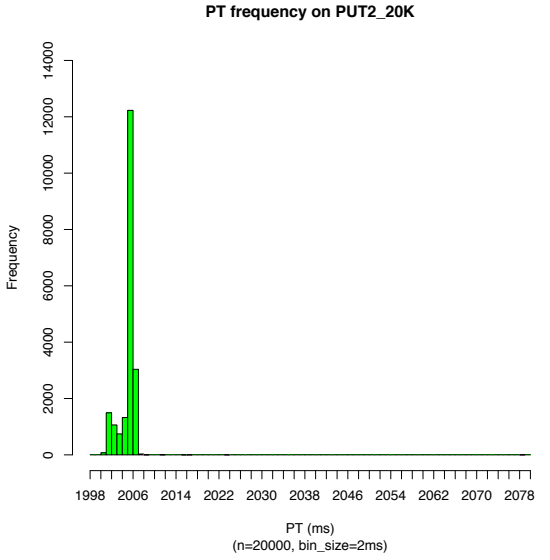
Table 5: The daemon processes captured at the hike of PUT2



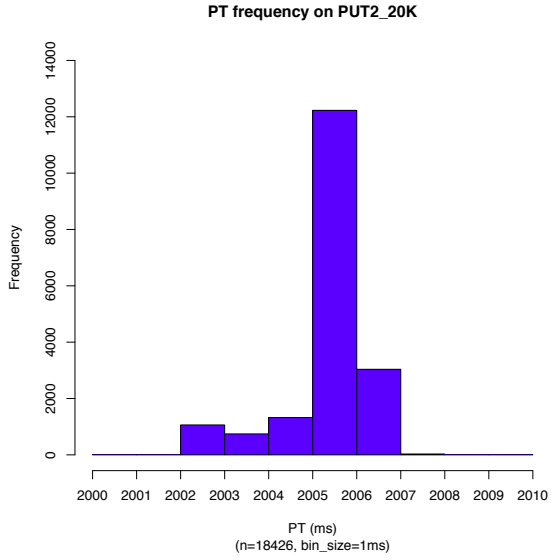
(a) PT frequency on PUT1 by EMPv4 (See Table 2.)



(b) PT frequency on PUT1 by EMPv5 (See Table 2.)

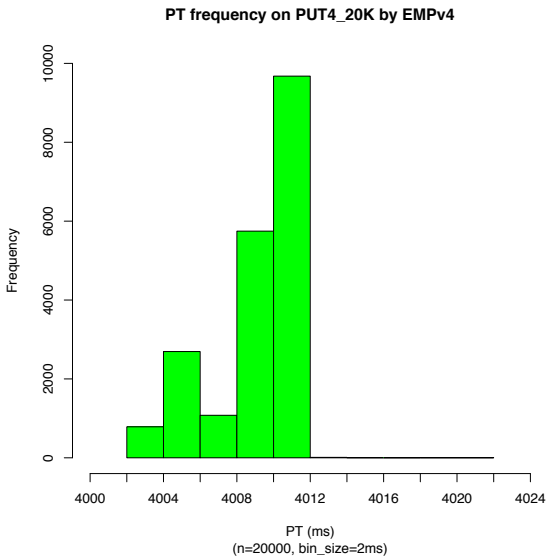


(c) PT frequency on PUT2 by EMPv4 (See Table 2.)

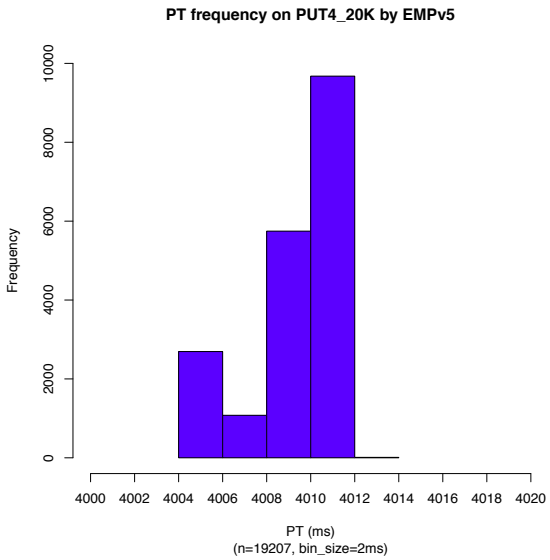


(d) PT frequency on PUT2 by EMPv5 (See Table 2.)

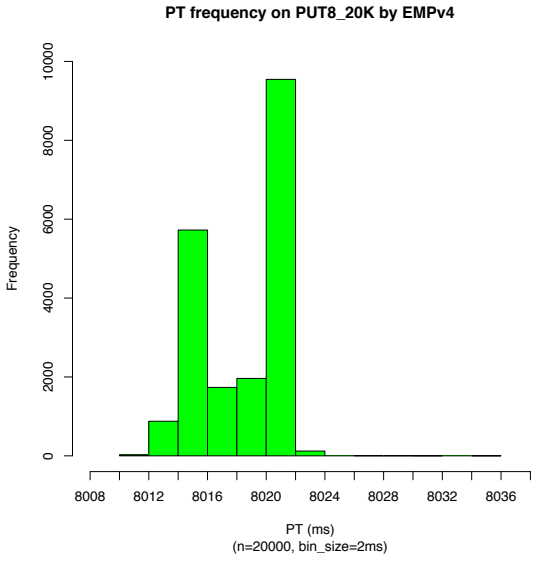
Figure 14: PT Histograms of PUT1 and PUT2 by 20,000 trials



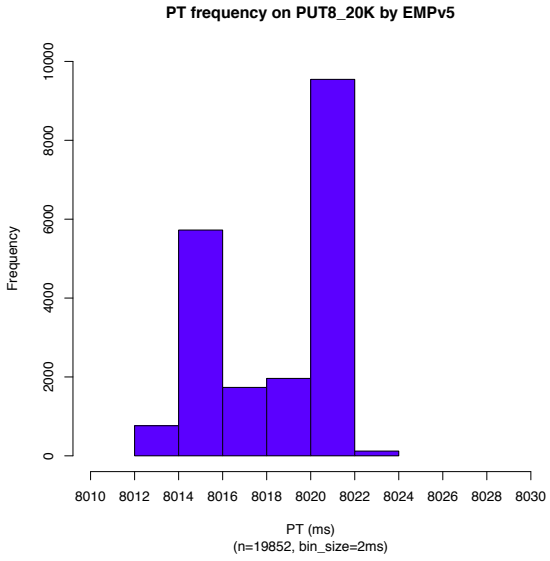
(a) PT frequency on PUT4 by EMPv4 (See Table 2.)



(b) PT frequency on PUT4 by EMPv5 (See Table 2.)



(c) PT frequency on PUT8 by EMPv4 (See Table 2.)



(d) PT frequency on PUT8 by EMPv5 (See Table 2.)

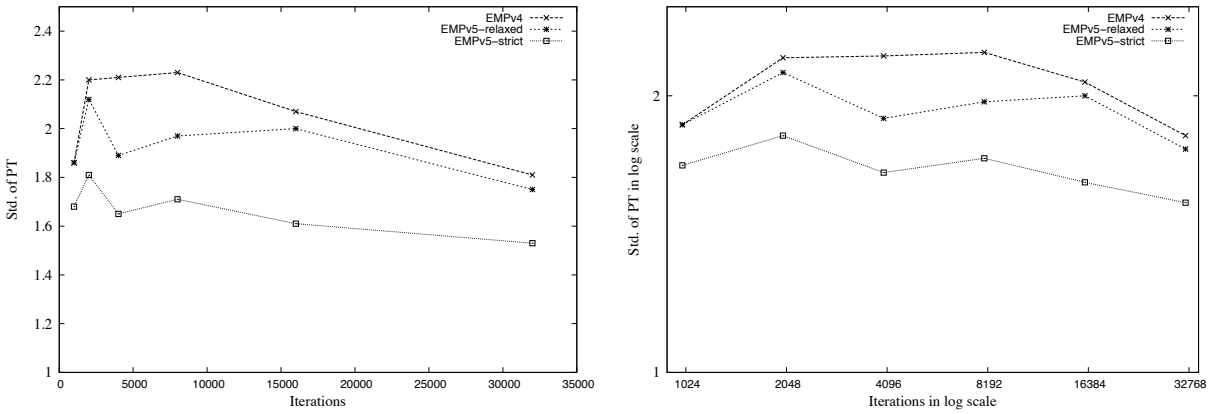
Figure 15: PT Histograms of PUT4 and PUT8 by 20,000 trials

5.2 PUT16

In this experiment we ran PUT16 up to 32,000 from 1,000 times by a factor of two. The relaxed version of EMPv5 (called *EMPv5-relaxed*) uses *five* standard deviations whereas its strict version (called *EMPv5-strict*) does *two* standard deviations for a vertical gap below and above the average. (Young: 2k samples seem most appropriate to represent the whole population of PUT16, in that the standard deviations by EMPv5 on the 2k sample size are almost at peak compared to those of the other sample sizes.)

Num. of Samples	Std. Dev. (msec)		
	EMPv4	EMPv5-relaxed	EMPv5-strict
1,000	1.86	1.86	1.68
2,000	2.20	2.12	1.81
4,000	2.21	1.89	1.65
8,000	2.23	1.97	1.71
16,000	2.07	2.00	1.61
32,000	1.81	1.75	1.53

Table 6: Standard deviations of PUT16 over increasing sample size



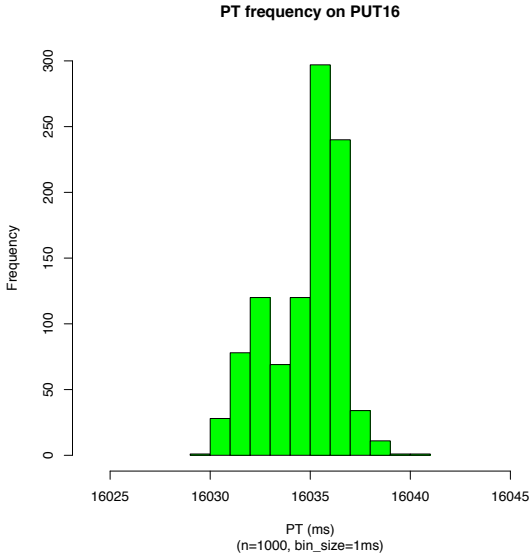
(a) Standard deviations of PT on PUT16

(b) Standard deviations of PT on PUT16, in log scale

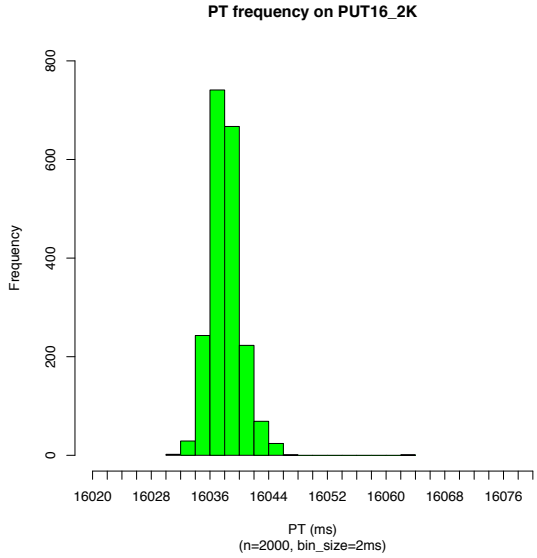
Figure 16: Standard deviations of PT on PUT16 over increasing sample size

5.3 Histograms by EMPv4

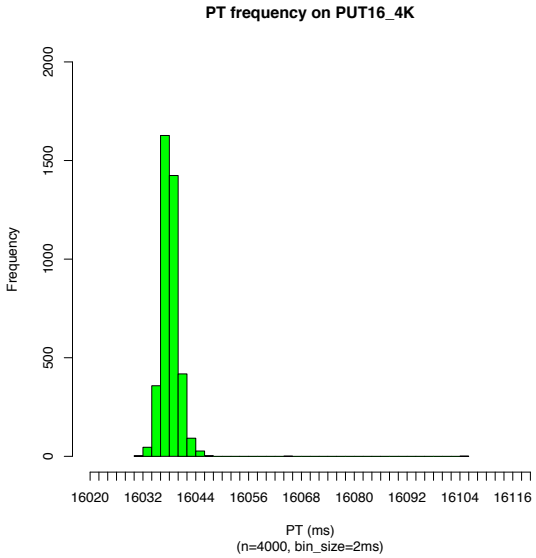
We apply EMPv4 to the runs of PUT16 as mentioned above. The following histograms are the results of EMPv4.



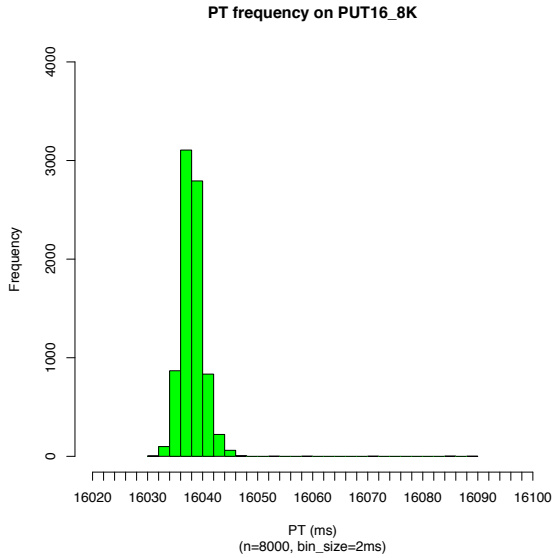
(a) PT frequency on PUT16 with 1k samples



(b) PT frequency on PUT16 with 2k samples

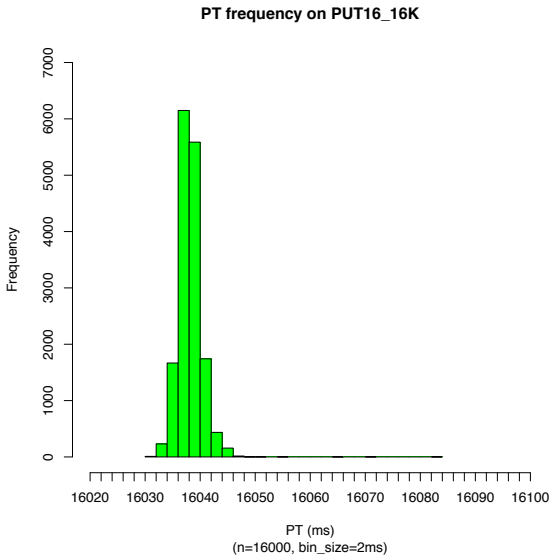


(c) PT frequency on PUT16 with 4k samples

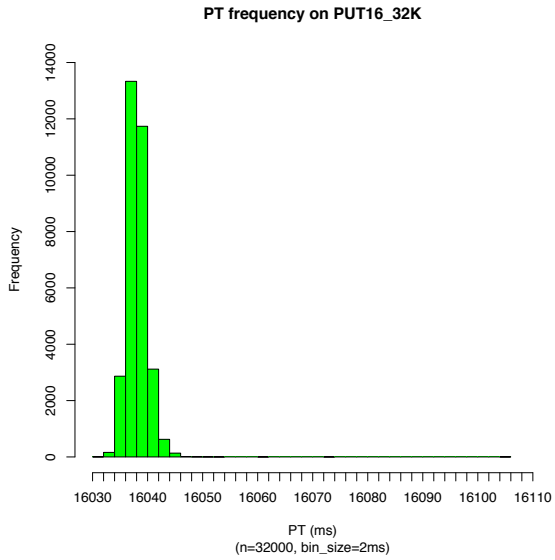


(d) PT frequency on PUT16 with 8k samples

Figure 17: PT histogram of PUT16 by EMPv4, with the sample size increasing from 1k to 8k



(a) PT frequency on PUT16 with 16k samples

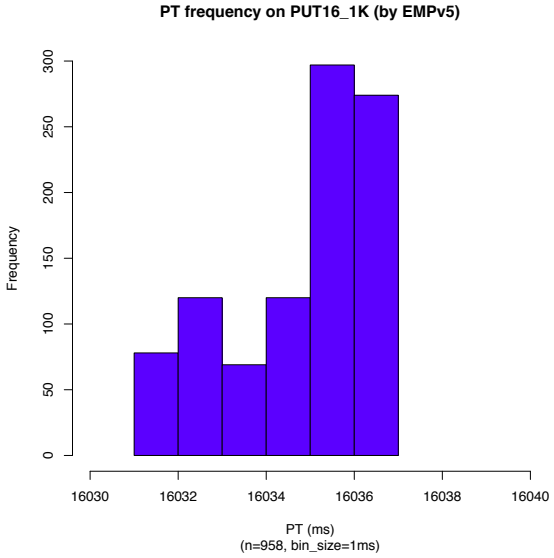


(b) PT frequency on PUT16 with 32k samples

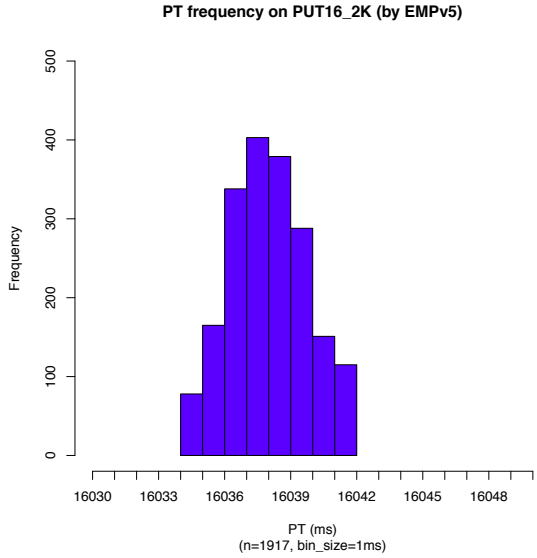
Figure 18: PT histogram of PUT16 by EMPv4, with the sample size increasing from 16k to 32k

5.4 Histograms by EMPv5

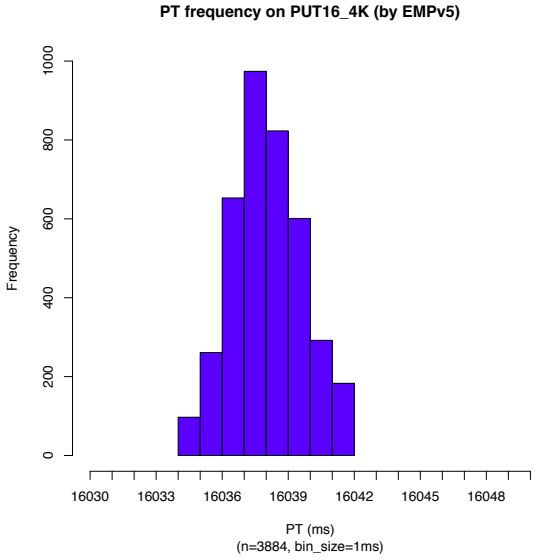
We now apply EMPv5 to the same data of PUT16. To be more specific, we use EMPv5-strict, by which the following histograms are obtained.



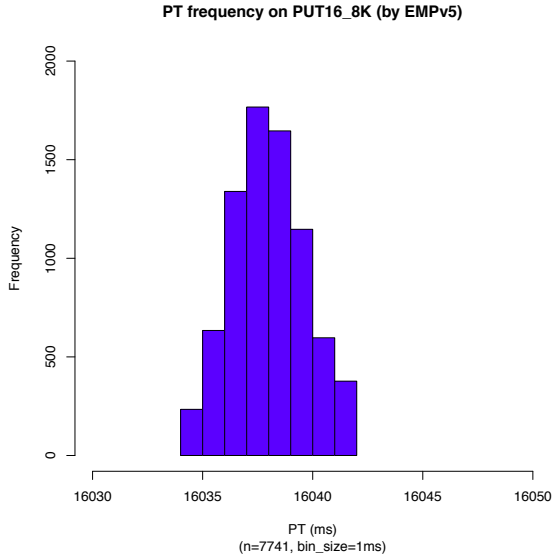
(a) PT frequency on PUT16 with 1k samples



(b) PT frequency on PUT16 with 2k samples

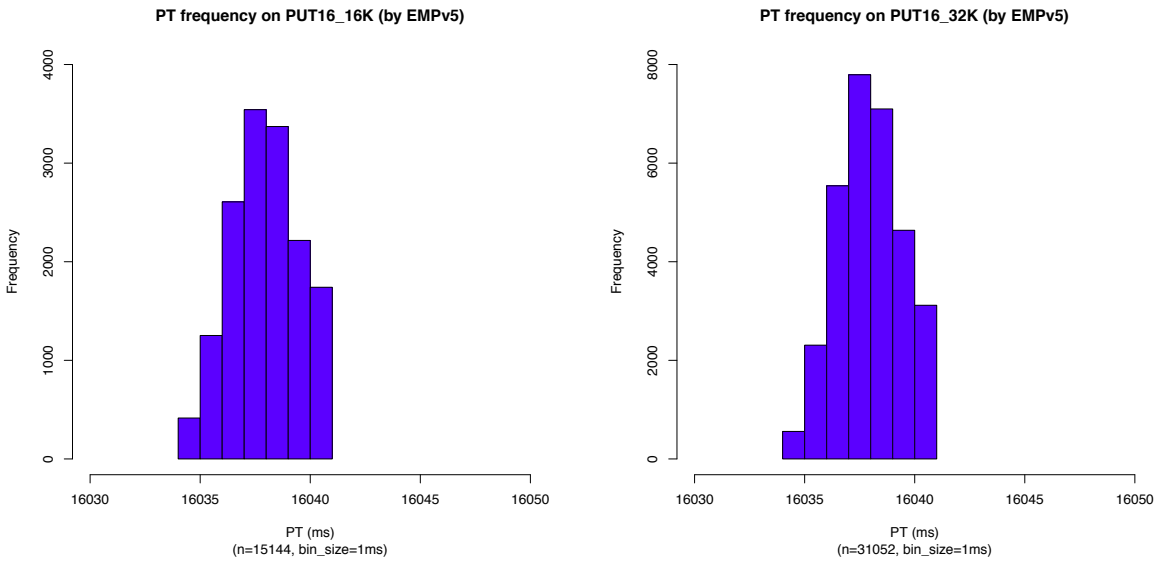


(c) PT frequency on PUT16 with 4k samples



(d) PT frequency on PUT16 with 8k samples

Figure 19: PT histogram of PUT16 by EMPv5, with the sample size increasing from 1k to 8k



(a) PT frequency on PUT16 with 16k samples

(b) PT frequency on PUT16 with 32k samples

Figure 20: PT histogram of PUT16 by EMPv5, with the sample size increasing from 16k to 32k

6 Dual PUT Experiment

In this section we study the characteristics of program times measured in the dual PUT experiments, in which the for-loop of PUT is broken up into two equal-sized for-loops (for_1 and for_2) of which elapsed times are individually measured using `gettimeofday()`. The experiment is designed to see whether there exists "internal dependency" of measured times of the two for-loops when PUT is timed. To be more specific, we compute correlation coefficients between corresponding measured times of for_1 and for_2, within the same run of each PUT. We also compute the correlation coefficients after removing some outliers determined by eye. In this experiment we expect that little dependency will be observed within the same run for any PUT. Note that except dual PUT4096 (on `sodb12`) all the other dual experiments were run on the same machine (`sodb9`), as described in Table 2. This could be one of possible reasons that the structure of dual PUT4096 looks quite different from that of the others although both `sodb9` and `sodb12` have the same machine specification.

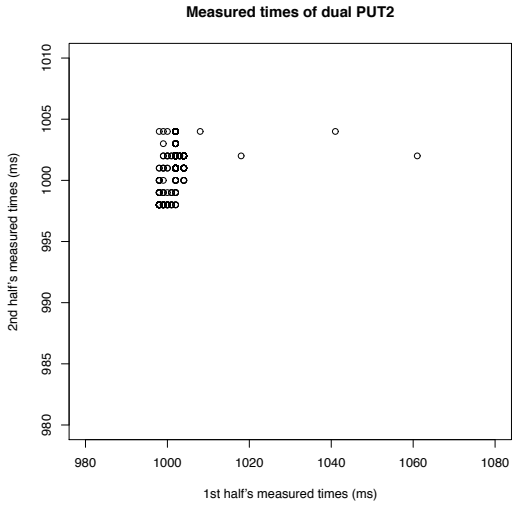
The base data of the following table and subsequent plots are from Table 2.

	Corr. Coeff.	Corr. Coeff. with outliers removed	Sample Size (# of regulars)
PUT2	0.3	0.5	1,000
PUT4	-0.07	-0.2	1,000
PUT8	0.8	-0.2	1,000
PUT16	0.3	-0.6	1,000
PUT32	0.01	-0.4	1,000
PUT64	0.003	-0.5	1,000
PUT128	0.04	-0.5	1,000
PUT256	0.004	-0.1	1,000
PUT512	-0.03	-0.1	1,000
PUT1024	0.14	-0.04	1,000
PUT2048	-0.01	-0.08	1,000
PUT4096	-0.01	-0.2	500

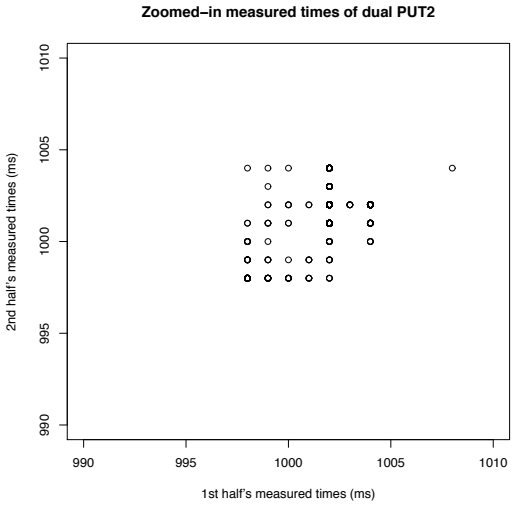
Table 7: Overall statistics of dual PUT experiment

6.1 Scatter Plots

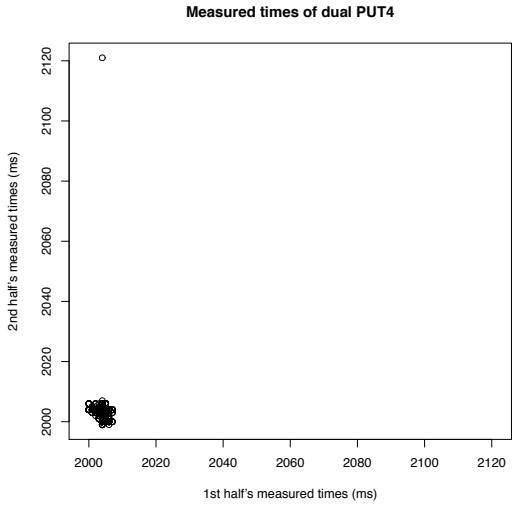
In this section we plot measured times of dual PUT experiments. We provide not only scatter plots of raw data but also those of focused clouds to further look inside. The focused clouds were obtained by cutting off outliers chosen by eye.



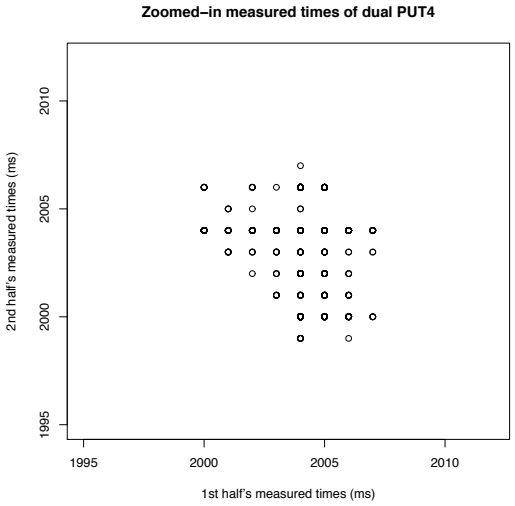
(a) Measured times on dual PUT2



(b) Zoomed-in measured times on dual PUT2

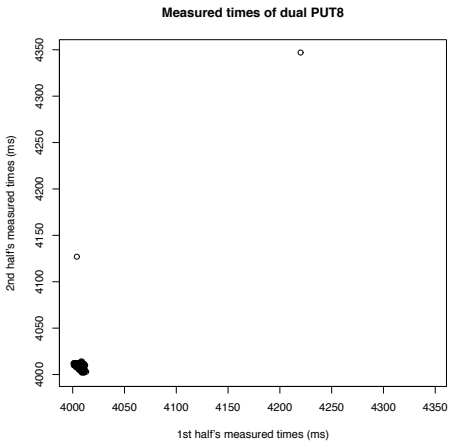


(c) Measured times on dual PUT4

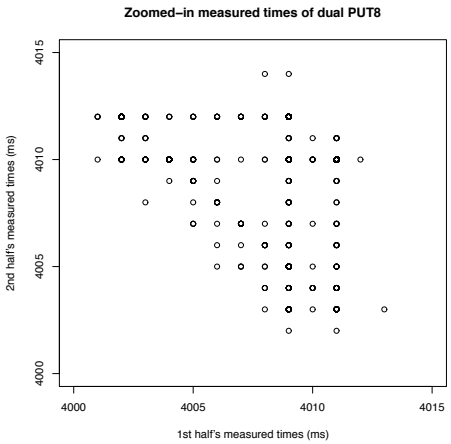


(d) Zoomed-in measured times on dual PUT4

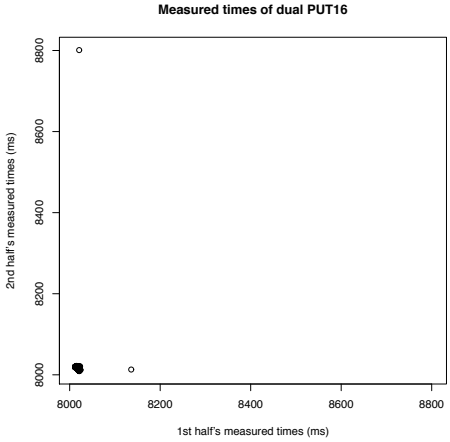
Figure 21: Scatter plots on dual PUT2∼PUT8



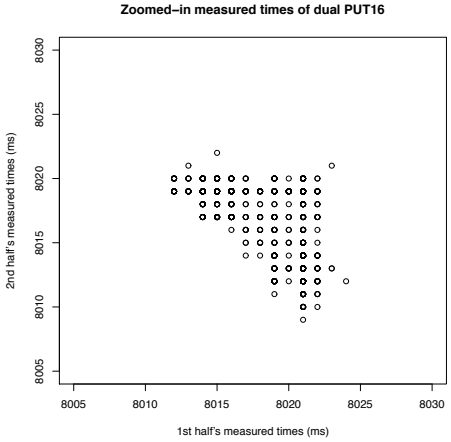
(a) Measured times on dual PUT8



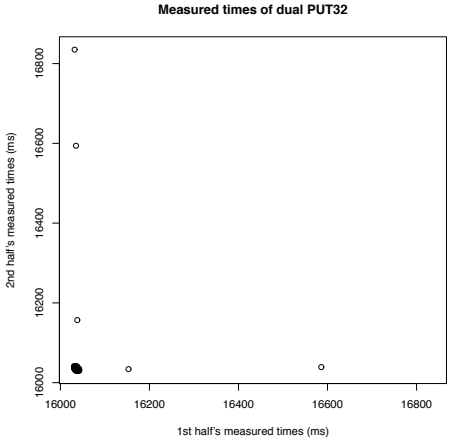
(b) Zoomed-in measured times on dual PUT8



(c) Measured times on dual PUT16



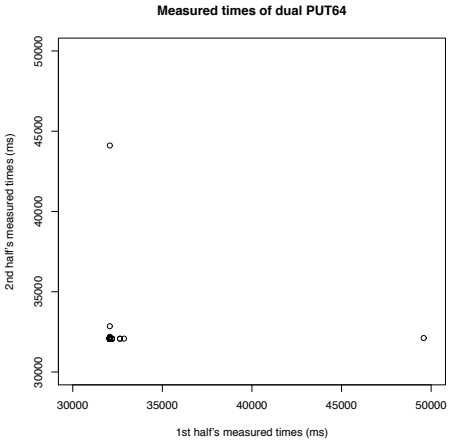
(d) Zoomed-in measured times on dual PUT16



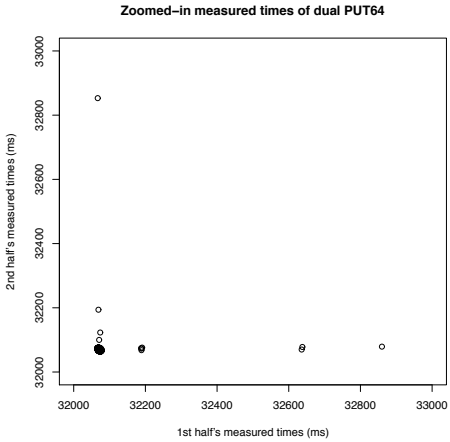
(e) Measured times on dual PUT32

(f) Zoomed-in measured times on dual PUT32

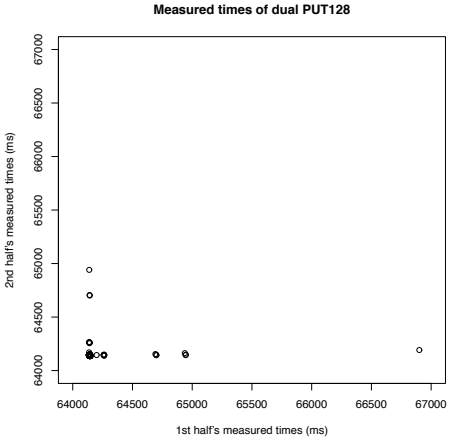
Figure 22: Scatter plots on dual PUT8∼PUT32



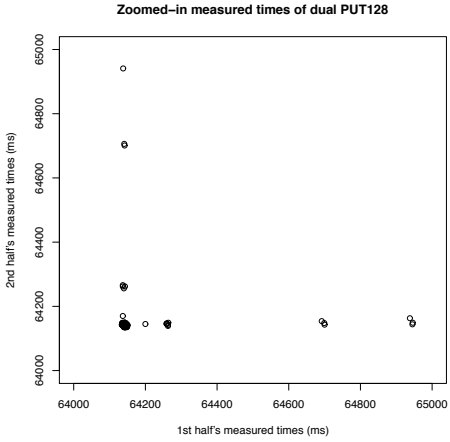
(a) Measured times on dual PUT64



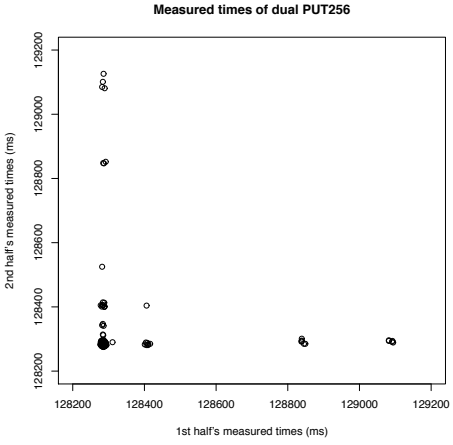
(b) Zoomed-in measured times on dual PUT64



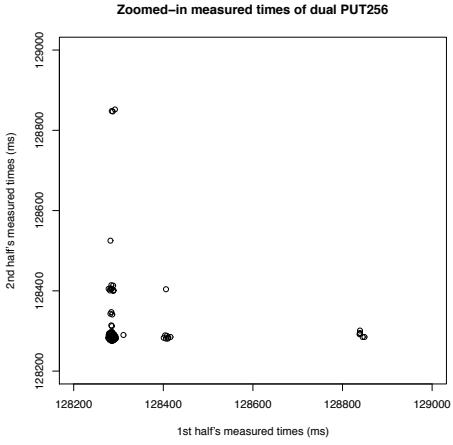
(c) Measured times on dual PUT128



(d) Zoomed-in measured times on dual PUT128

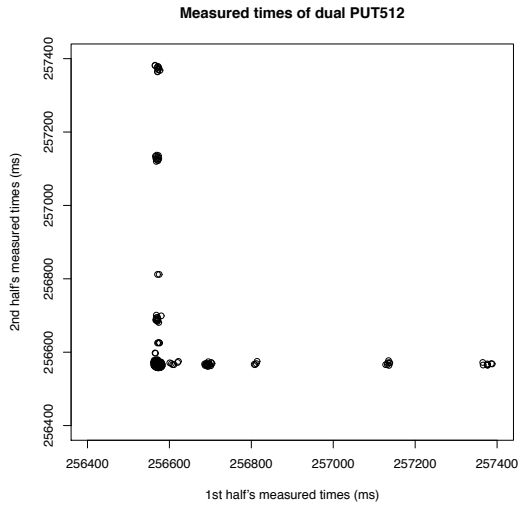


(e) Measured times on dual PUT256

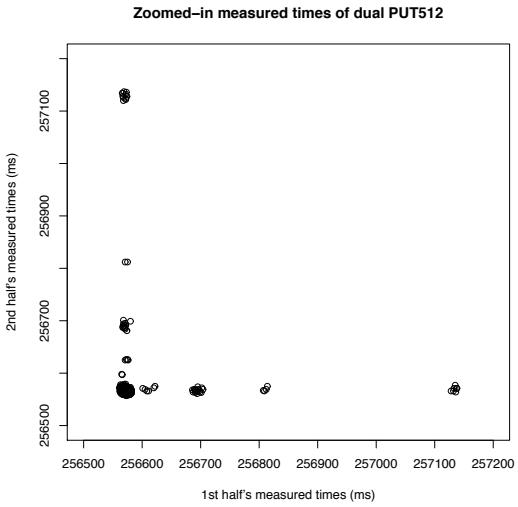


(f) Zoomed-in measured times on dual PUT256

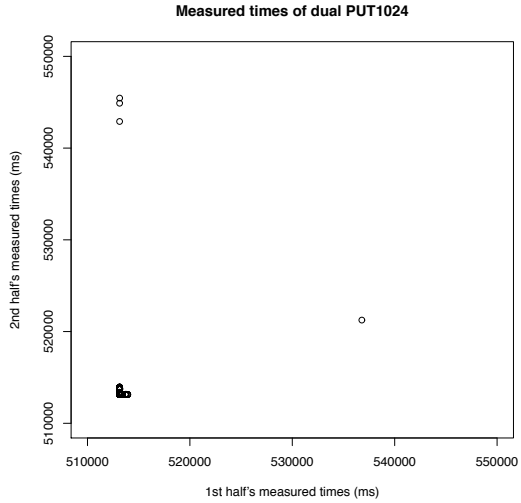
Figure 23: Scatter plots on dual PUT64∼PUT256



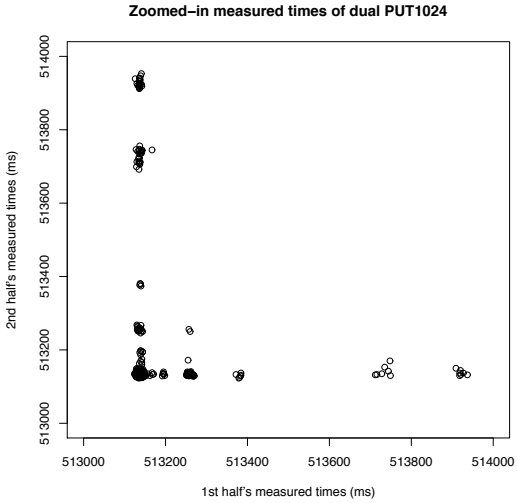
(a) Measured times on dual PUT512



(b) Zoomed-in measured times on dual PUT512

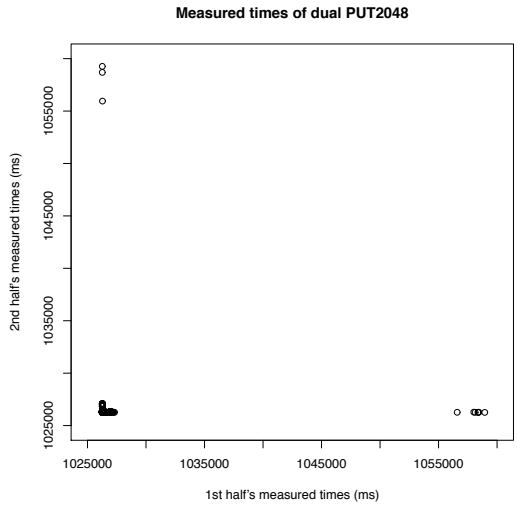


(c) Measured times on dual PUT1024

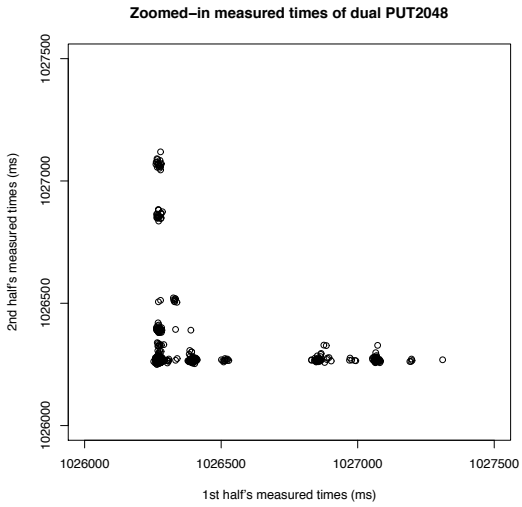


(d) Zoomed-in measured times on dual PUT1024

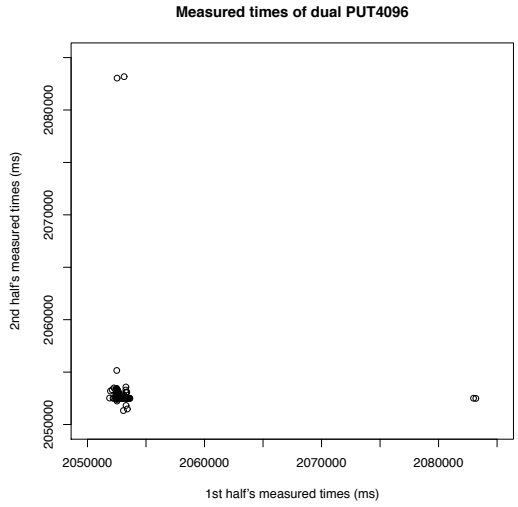
Figure 24: Scatter plots on dual PUT512 and PUT1024



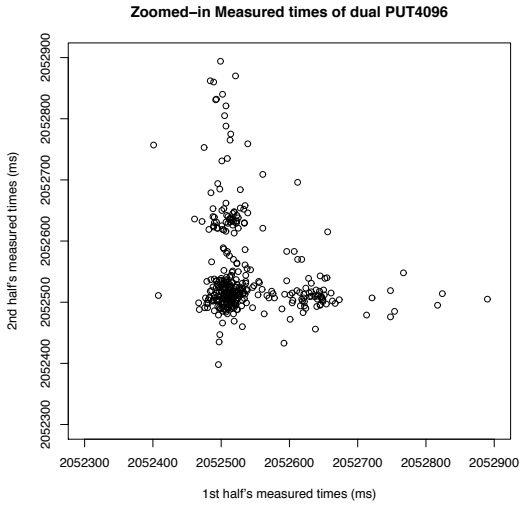
(a) Measured times on dual PUT2048



(b) Zoomed-in measured times on dual PUT2048



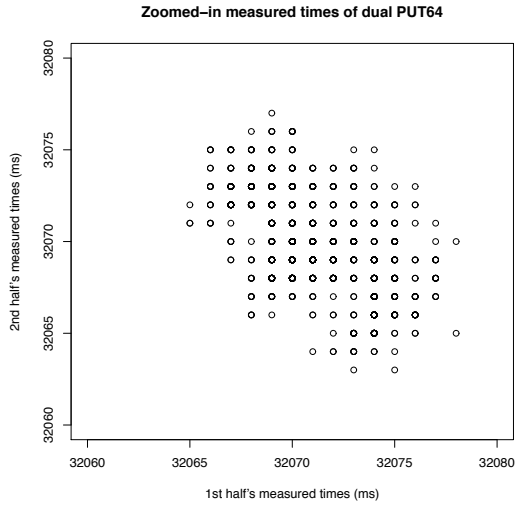
(c) Measured times on dual PUT4096



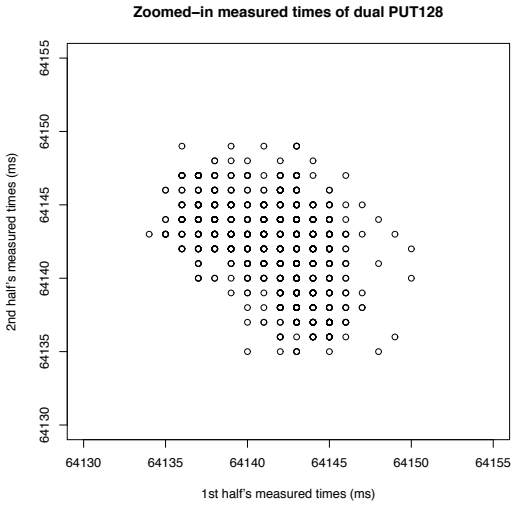
(d) Zoomed-in measured times on dual PUT4096

Figure 25: Scatter plots on dual PUT2048∼PUT4096

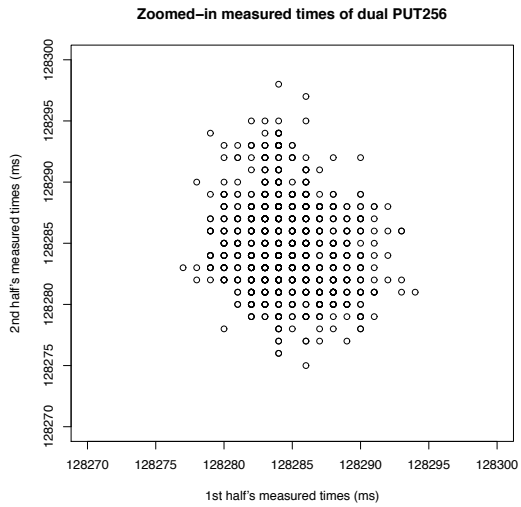
6.1.1 Supplementary Scatter Plots



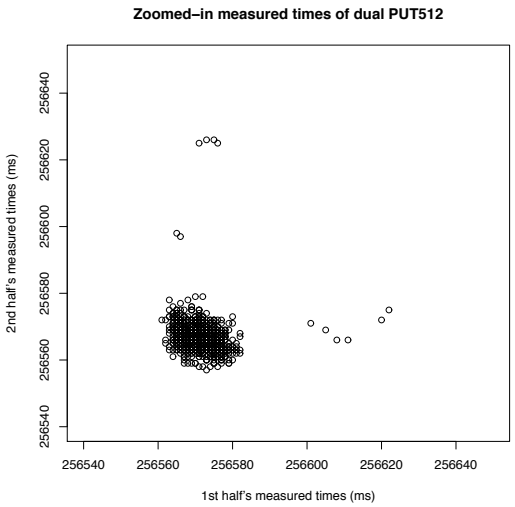
(a) The central clump of dual PUT64



(b) The central clump of dual PUT128

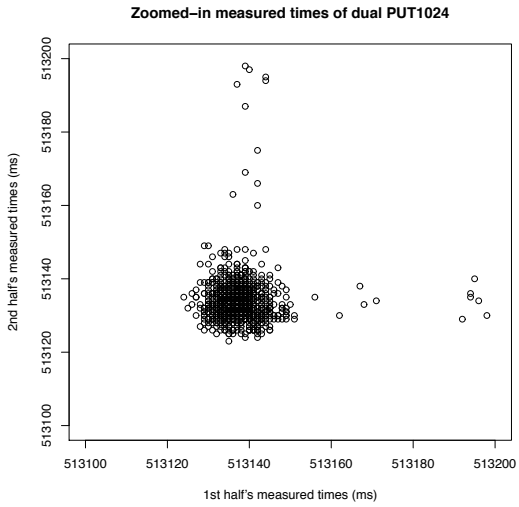


(c) The central clump of dual PUT256

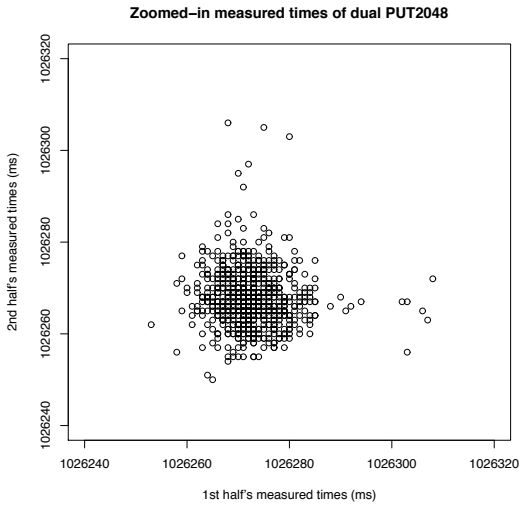


(d) The central clump of dual PUT512

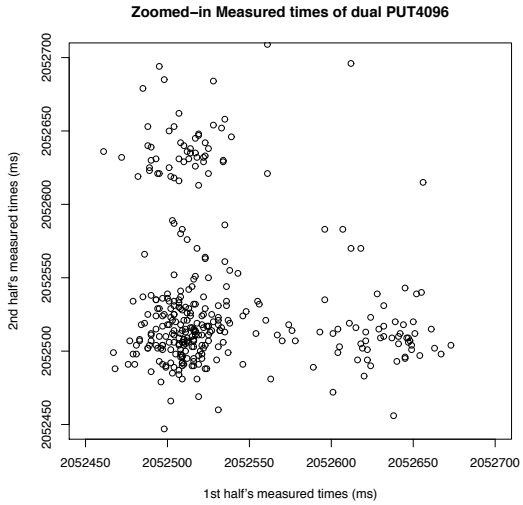
Figure 26: Focused clumps - part I



(a) The central clump of dual PUT1024



(b) The central clump of dual PUT2048



(c) The central clump of dual PUT4096

Figure 27: Focused clumps - part II

6.1.2 Captured Processes on Dual PUT4096

In this study we examine a list of processes captured in a specific region of measured times on dual PUT4096. The base data is from Figure 25(c). We divide the data into four subregions by the pivot point of "2,052,575" msec on the x and y axes. This pivot is chosen by eye, based on Figure 27(c).

Table 8 represents a list of daemon processes captured in each of the four subregions while Table 9 shows a list of regions in which a specific daemon process(es) appeared. The last column in Table 8 indicates the average time executed by the captured daemon processes in each region. Table 9 also shows how long each daemon process was on average run whenever it appeared in a specific region.

Subregion	Daemon Processes	Avg. Daemon (Running) Time (msecs)	# of Samples
Region 1 ($x <$ pivot, $y <$ pivot)	`bash`, `id`, `java`, `rhn_check`, `rhsmcertd-worke`, `sshd`, `uname`	3,494	214
Region 2 ($x <$ pivot, $y \geq$ pivot)	`id`, `java`, `rhn_check`, `rhsmcertd-worke`, `sshd`	6,954	137
Region 3 ($x \geq$ pivot, $y <$ pivot)	`java`, `rhn_check`, `rhsmcertd-worke`, `sshd`	16,350	128
Region 4 ($x \geq$ pivot, $y \geq$ pivot)	`java`, `rhn_check`, `rhsmcertd-worke`, `sshd`	2,130	21

Table 8: Captured Daemon Processes and Their Times in Each Region

Daemon Processes	Captured Region List
`bash`	1 (370 msecs)
`id`	1 (10 msecs), 2 (10 msecs)
`java`	1 (11 msecs), 2 (11 msecs), 3 (12 msecs), 4 (12 msecs)
`rhn_check`	1 (6,415 msecs), 2 (25,855 msecs), 3 (21,923 msecs), 4 (7,585 msecs)
`rhsmcertd-worke`	1 (1,157 msecs), 2 (1,156 msecs), 3 (1,158 msecs), 4 (1,152 msecs)
`sshd`	1 (183 msecs), 2 (64 msecs), 3 (80 msecs), 4 (75 msecs)
`uname`	1 (10 msecs)

Table 9: Per-Daemon Appearance Region and Its Averge Running Time

6.1.3 Analysis of Outliers on Dual PUT128

An in-depth analysis of outliers in Figure 23(c) was conducted. This analysis concerns what daemon processes with positive CPU time were captured and how much time was taken by them. The value in the parentheses is obtained when `proc_monitor` is considered.

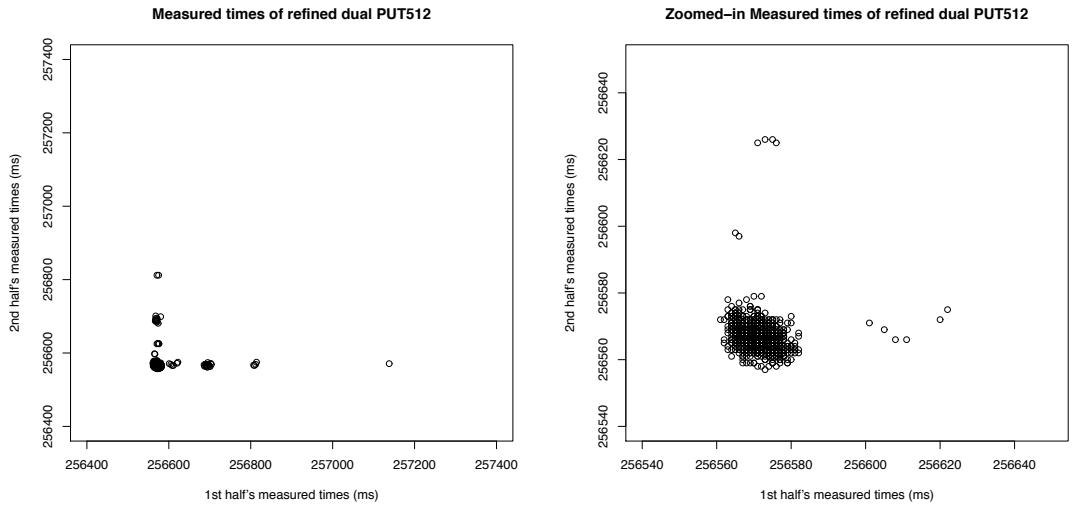
Iter. #	1st Half (msec)	2nd Half (msec)	Daemon Procs.	Daem. Time (msec)
68	64142	64701	`kslowd000`, `kslowd001`, `md0_raid1`, `java` (, `proc_monitor`)	18 (220)
180	64138	64941	`kslowd000`, `kslowd001`, `md0_raid1`, `java` (, `proc_monitor`)	20 (224)
292	64141	64706	`kslowd000`, `kslowd001`, `md0_raid1`, `java` (, `proc_monitor`)	19 (221)
351	64938	64163	`kblockd/0`, `kslowd000`, `kslowd001`, `md0_raid1`, `java` (, `proc_monitor`)	19 (221)
435	64692	64154	`kslowd000`, `kslowd001`, `md0_raid1`, `java` (, `proc_monitor`)	17 (219)
547	66901	64192	`kslowd000`, `kslowd001`, `java` (, `proc_monitor`)	16 (218)
659	64699	64148	`kslowd000`, `kslowd001`, `md0_raid1`, `java` (, `proc_monitor`)	19 (221)
771	64946	64149	`kslowd000`, `kslowd001`, `md0_raid1`, `java` (, `proc_monitor`)	18 (220)
883	64700	64143	`kslowd000`, `kslowd001`, `md0_raid1`, `java` (, `proc_monitor`)	17 (219)
986	64945	64144	`kslowd000`, `kslowd001`, `java` (, `proc_monitor`)	16 (218)
Corr. eff. b/w PUT and daem. proc. times	-0.67 (-0.63)			

Table 10: Further Examination of Outliers on Dual PUT128 in Figure 23(c)

6.1.4 Refined Dual PUT Data

We conduct another in-depth study on some outliers of the dual PUT512, PUT1024, PUT2048 and PUT4096 data. The outliers are considered the ones with a daemon process(es) (e.g.,`rhn_check`) run for a significant amount of time (more than 500 msecs for dual PUT512 and 1,000 msecs for the other three, respectively).

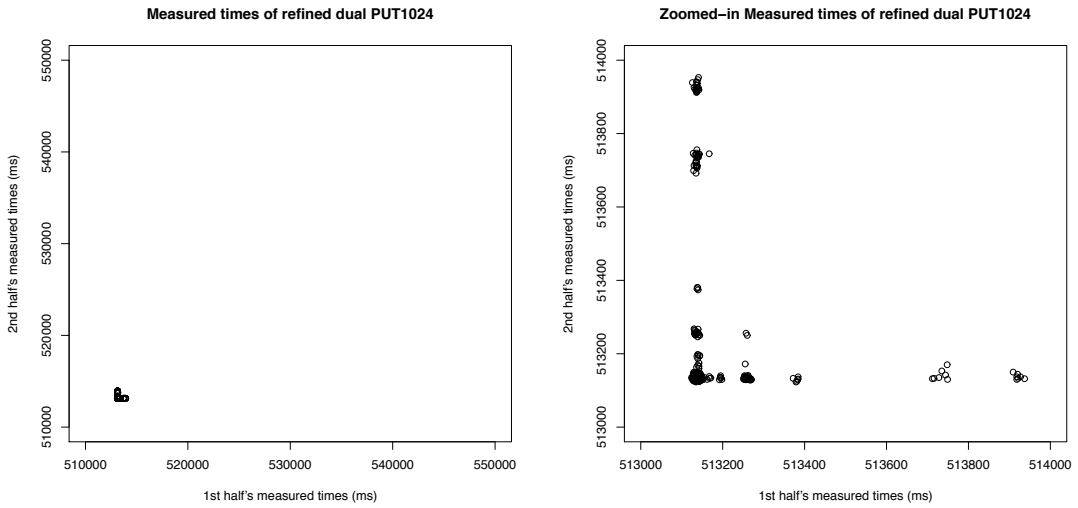
Dual 512: In Figure 28, a total of 36 points are identified as outliers and removed, compared to that of Figure 24(a).



(a) Refined Dual PUT1512 (b) The Focused Clump of Refined Dual PUT512

Figure 28: Refined Dual PUT512 with Significant `rhn_check` Removed

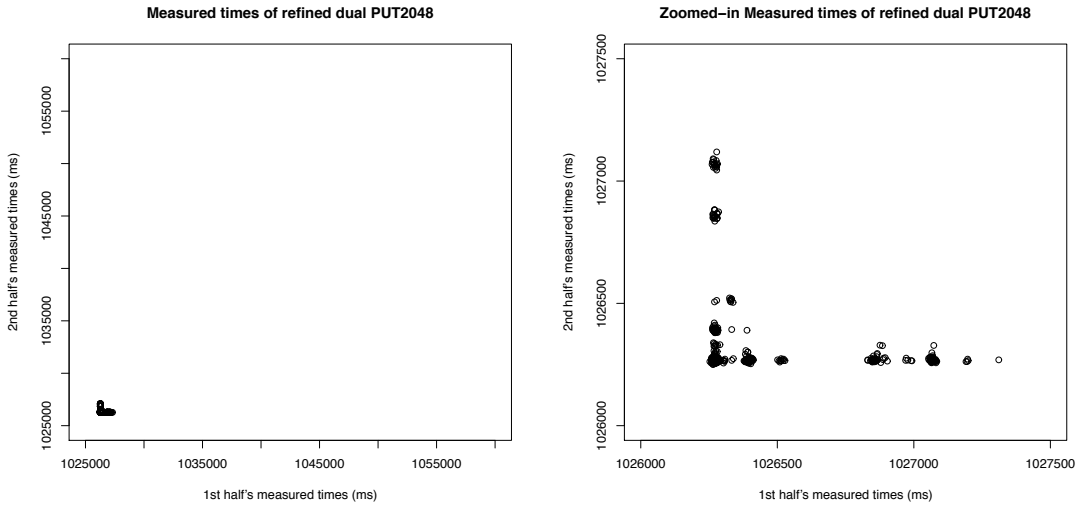
Dual 1024: In Figure 29(a), just four points are removed compared to that of Figure 24(c).



(a) Refined Dual PUT1024 (b) The Focused Clump of Refined Dual PUT1024

Figure 29: Refined Dual PUT1024 with Significant `rhn_check` Removed

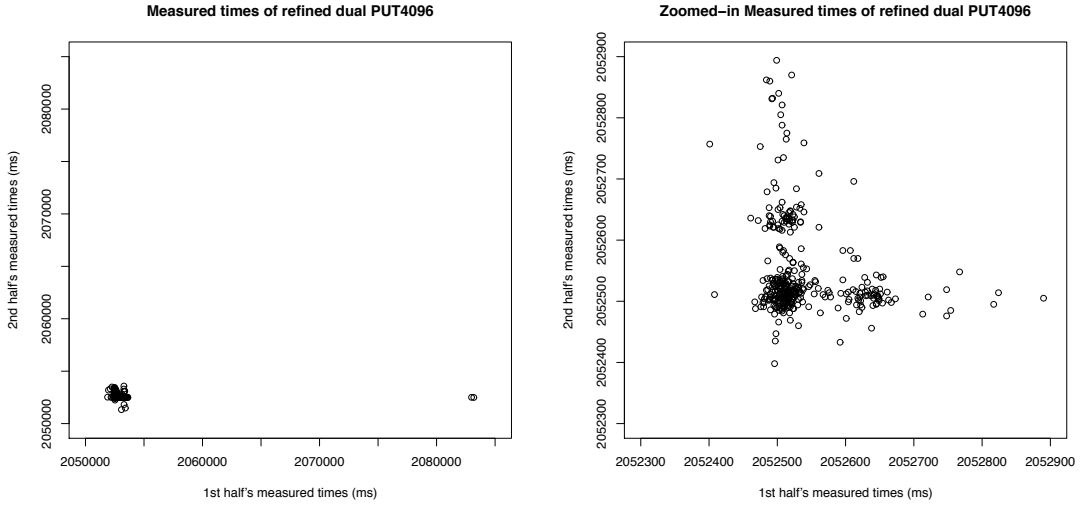
Dual 2048: In Figure 30(a), ten points are removed compared to that of Figure 25(a).



(a) Refined Dual PUT2048 (b) The Focused Clump of Refined Dual PUT2048

Figure 30: Refined Dual PUT2048 with Significant `rhn_check` Removed

Dual 4096: See Table 2 for the details of this run (on `sodb8`). In Figure 31(a), five points are removed compared to that of Figure 25(c).



(a) Refined Dual PUT4096 (b) The Focused Clump of Refined Dual PUT4096

Figure 31: Refined Dual PUT4096 with Significant `rhn_check` Removed

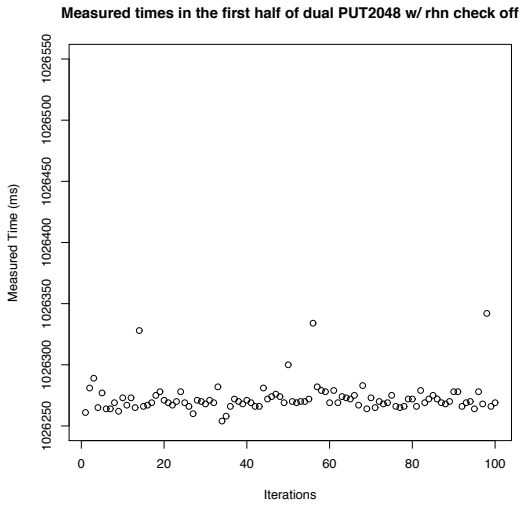
6.1.5 Dual PUT Data with `rhn_check` Disabled

In this experiment we switched off the `rhn_check` daemon and then ran dual PUT2048 (on `sodb9`) and PUT4096 (on `sodb8`) 100 times. For more details of the two runs, see Table 2. Table 11 compares the measurement quality of each half when `rhn_check` was enabled and disabled.

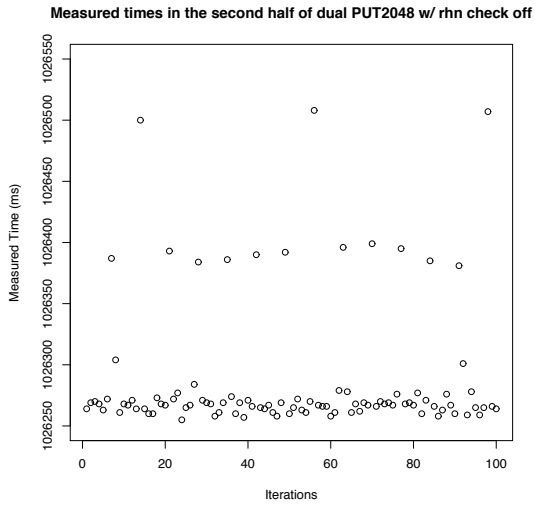
	Sample Size	Std. Dev. (msec) in 1st	Std. Dev. (msec) in 2nd	Corr. Coeff.
Dual PUT2048 w/ `rhn_check`	1000	2659.249	1739.93	-0.008
Dual PUT2048 w/o `rhn_check`	100	12.66	54.79	0.55
Dual PUT4096 w/ `rhn_check`	500	1942.12	1946.38	-0.005
Dual PUT4096 w/o `rhn_check`	100	47.26	73.49	-0.22

Table 11: Standard deviations of dual PUT2048 and PUT4096

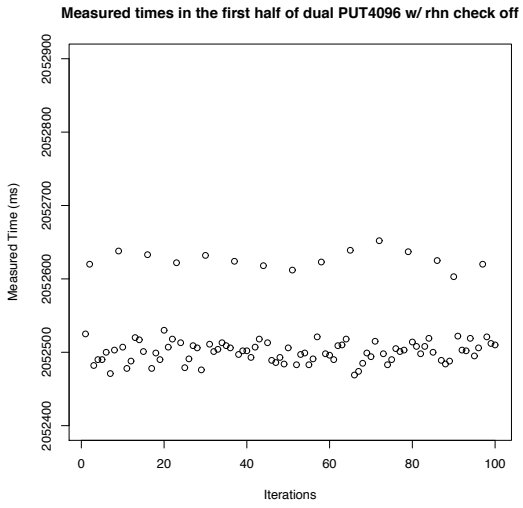
Figure 32 compares the measurements of the first and second halves of the dual PUTs.



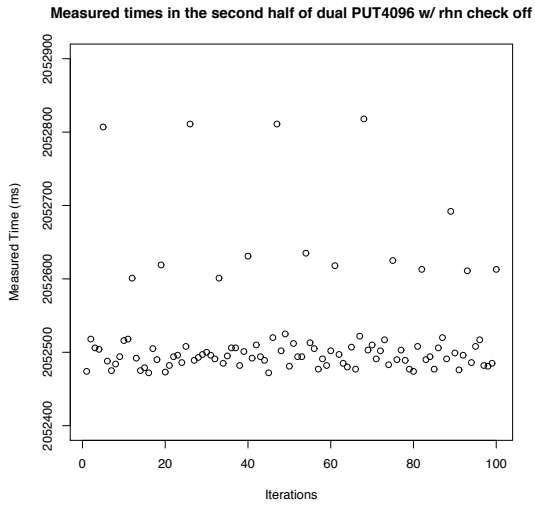
(a) 1st half's measured times on dual PUT2048



(b) 2nd half's measured times on dual PUT2048



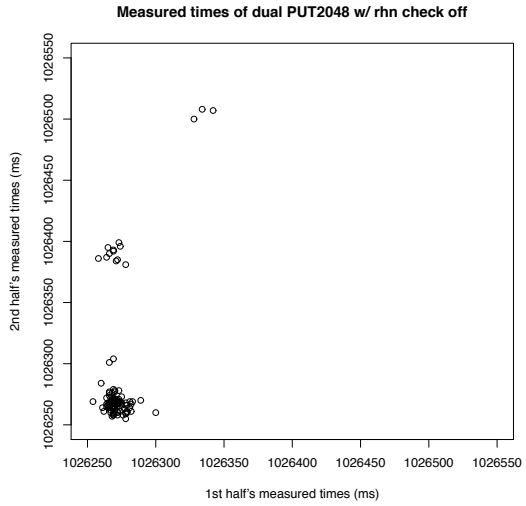
(c) 1st half's measured times on dual PUT4096



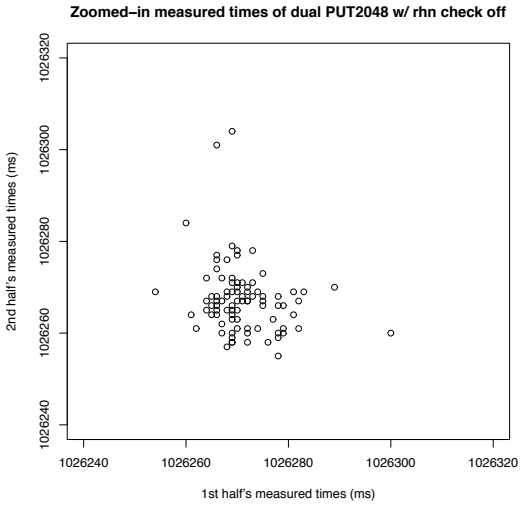
(d) 2nd half's measured times on dual PUT4096

Figure 32: Comparison of measured times on dual PUT2048 and PUT4096 with `rhn_check` off

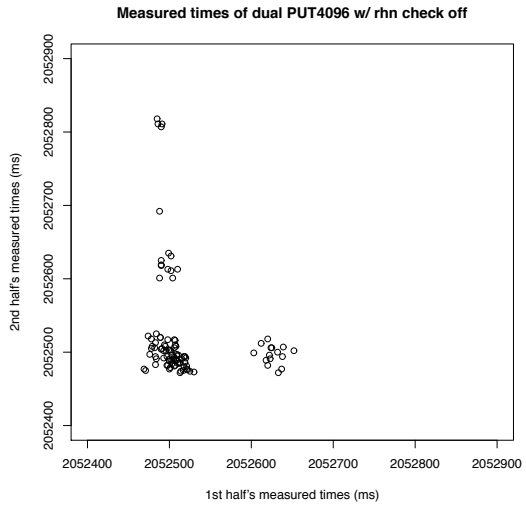
Figure 33 shows raw and zoomed-in scatter plots of the measurements of the dual PUTs.



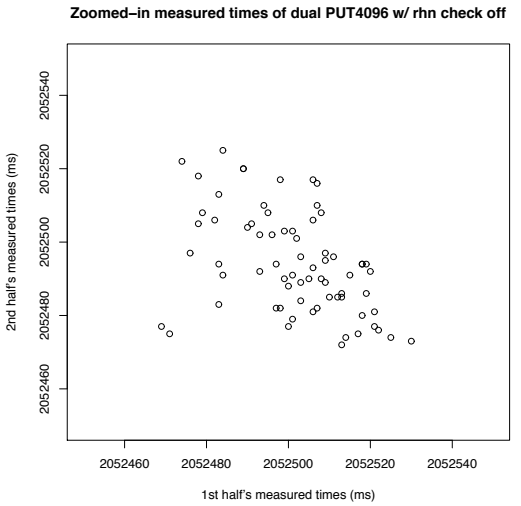
(a) Measured times on dual PUT2048



(b) Zoomed-in measured times on dual PUT2048



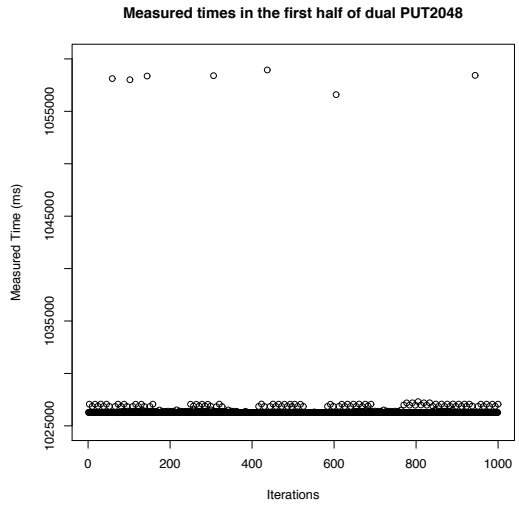
(c) Measured times on dual PUT4096



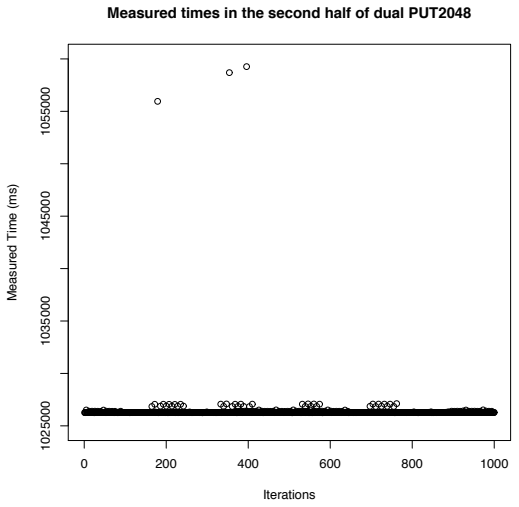
(d) Zoomed-in measured times on dual PUT4096

Figure 33: Scatter plots on dual PUT2048 and PUT4096 with `rhn_check` off

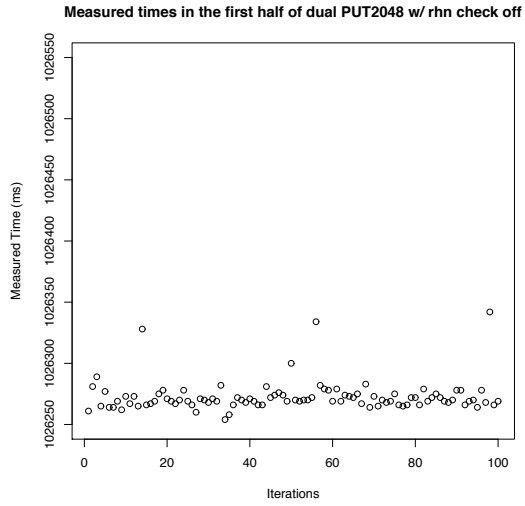
6.1.6 Influence of `rhn_check` on the Same Dual PUT



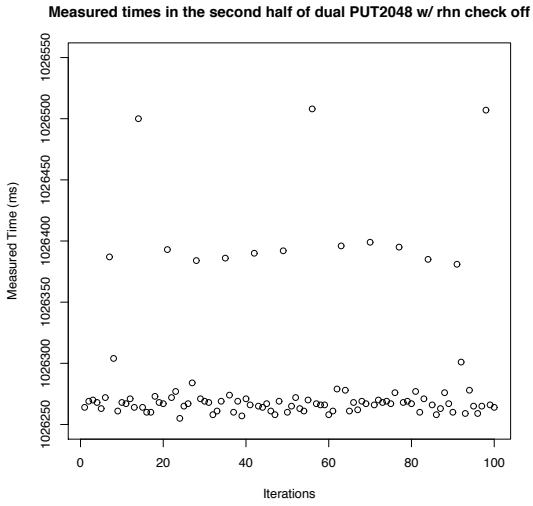
(a) 1st half's measured times on dual PUT2048 with `rhn_check` on



(b) 2nd half's measured times on dual PUT2048 with `rhn_check` on

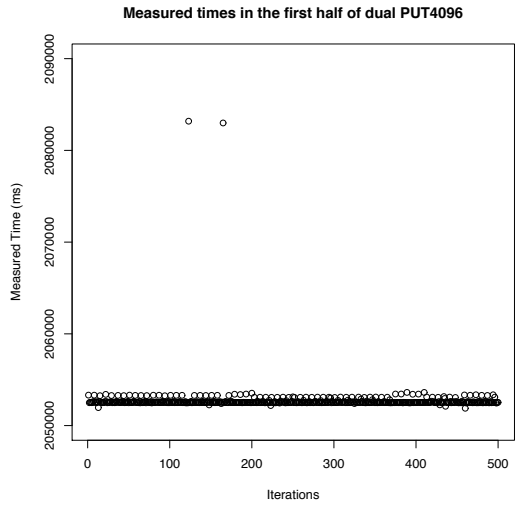


(c) 1st half's measured times on dual PUT2048 with `rhn_check` off

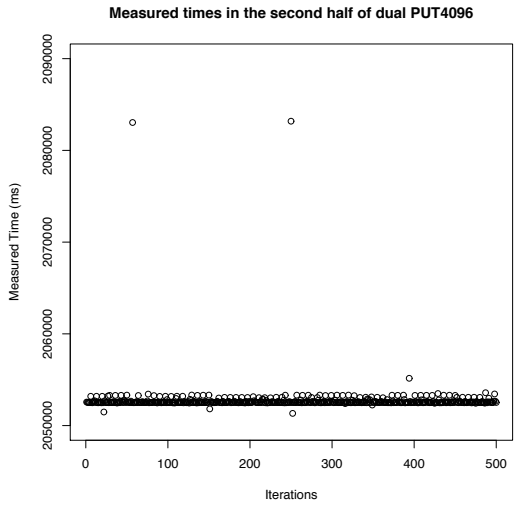


(d) 2nd half's measured times on dual PUT2048 with `rhn_check` off

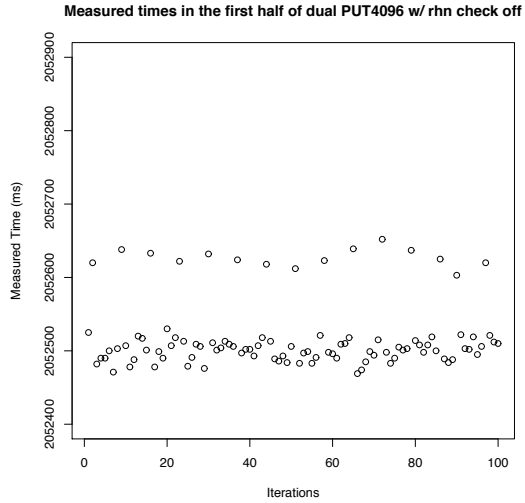
Figure 34: Comparison of measured times on dual PUT2048 with `rhn_check` on and off



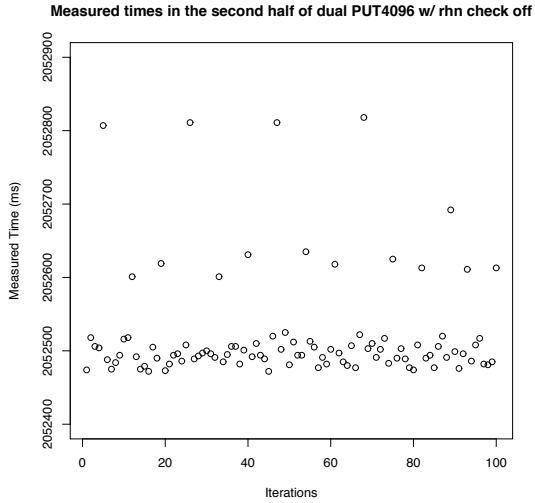
(a) 1st half's measured times on dual PUT4096 with `rhn_check` on



(b) 2nd half's measured times on dual PUT4096 with `rhn_check` on



(c) 1st half's measured times on dual PUT4096 with `rhn_check` off



(d) 2nd half's measured times on dual PUT4096 with `rhn_check` off

Figure 35: Comparison of measured times on dual PUT4096 with `rhn_check` on and off

6.1.7 Captured Daemon Processes at Some Outliers

Dual 4096 with `rhn_check` active: Data based on Figure 25(c).

Iter. #	Process (Time (msec))
4	`rhn_check` (30031)
249	`rhn_check` (30111)

Table 12: Captured daemon processes at outliers on dual PUT4096

Dual 4096 with `rhn_check` removed: Data based on Figure 33(c).

Iter. #	Process (Time (msec))
4	`sshd` (28), `grep` (9), `rhsmcertd-worke` (115), `rhsmcertd-worke` (115)
25	`sshd` (27), `grep` (8), `rhsmcertd-worke` (114), `rhsmcertd-worke` (115)
46	`sshd` (26), `grep` (10), `rhsmcertd-worke` (115), `rhsmcertd-worke` (115)
67	`sshd` (29), `grep` (11), `java` (1), `rhsmcertd-worke` (116), `rhsmcertd-worke` (115)

Table 13: Captured daemon processes at outliers on dual PUT4096

Dual 2048 with `rhn_check` active: Data based on Figure 25(a).

Iter. #	Process (Time (msec))
178	`rhn_check` (29211), `flush-9:0` (112), `kslowd001` (106), `md0_raid1` (106), `kslowd000` (104), `jbd2/md0-8` (20), `rhnsd` (6), `java` (4), `cifsd` (1)
353	`rhn_check` (31877), `flush-9:0` (146), `kslowd001` (105), `kslowd000` (104), `md0_raid1` (99), `jbd2/md0-8` (15), `rhnsd` (6), `java` (3), `cifsd` (1)
395	`rhn_check` (32436), `flush-9:0` (142), `md0_raid1` (116), `kslowd001` (105), `kslowd000` (104), `jbd2/md0-8` (11), `rhnsd` (4), `java` (3), `cifsd` (2), `kblockd/0` (1)

Table 14: Captured daemon processes at outliers on dual PUT2048

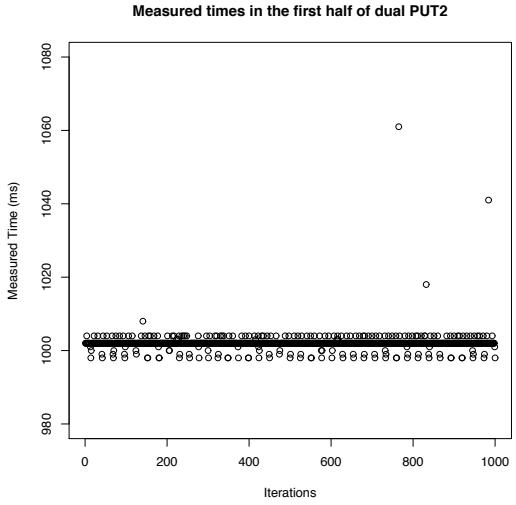
Dual 2048 with `rhn_check` removed: Data based on Figure 33(a).

Iter. #	Process (Time (msec))
13	`rhsmcertd-worke` (234), `kslowd000` (103), `kslowd001` (102), `ssd` (27), `md0_raid1` (26), `grep` (16), `java` (2), `jbd2/md0-8` (1)
55	`rhsmcertd-worke` (232), `kslowd000` (103), `kslowd001` (102), `ssd` (26), `md0_raid1` (26), `grep` (8), `kblockd/0` (2), `java` (2), `jbd2/md0-8` (1)
99	`rhsmcertd-worke` (233), `kslowd000` (103), `kslowd001` (102), `ssd` (33), `md0_raid1` (27), `grep` (10), `java` (2), `flush-9:0` (1) `kblockd/0` (1)

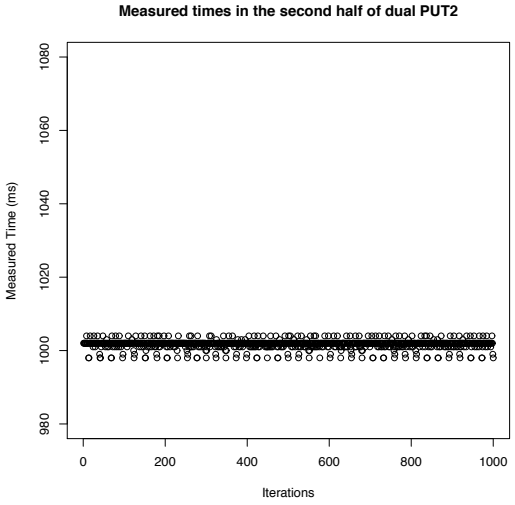
Table 15: Captured daemon processes at outliers on dual PUT2048

6.2 Program Time Comparison

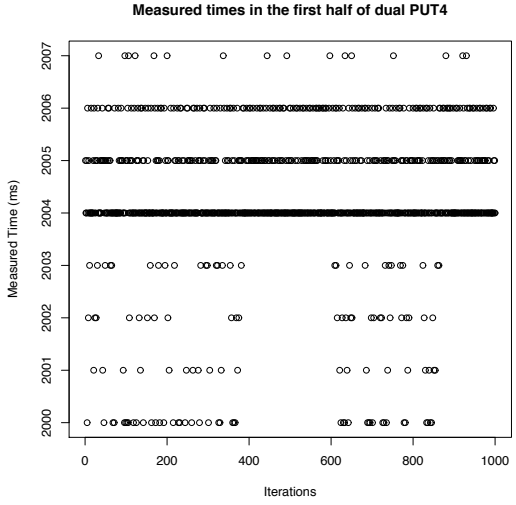
In this section we perform one-to-one comparison on measured times of parts I and II for the same iteration of each PUT.



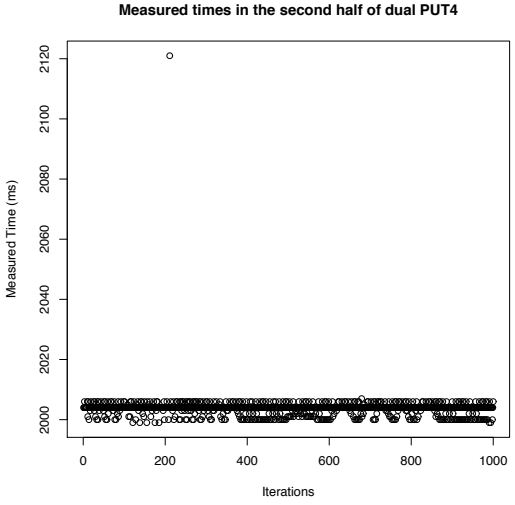
(a) 1st half's measured times on dual PUT2



(b) 2nd half's measured times on dual PUT2

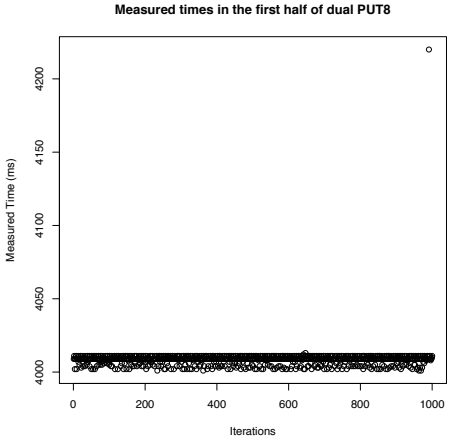


(c) 1st half's measured times on dual PUT4

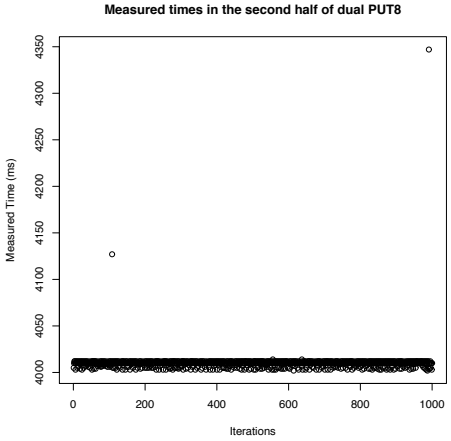


(d) 2nd half's measured times on dual PUT4

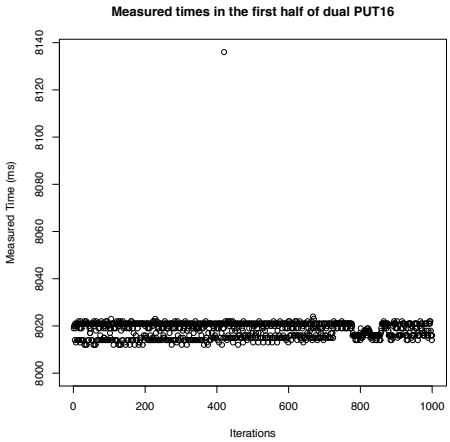
Figure 36: Comparison of measured times on dual PUT2∼PUT4



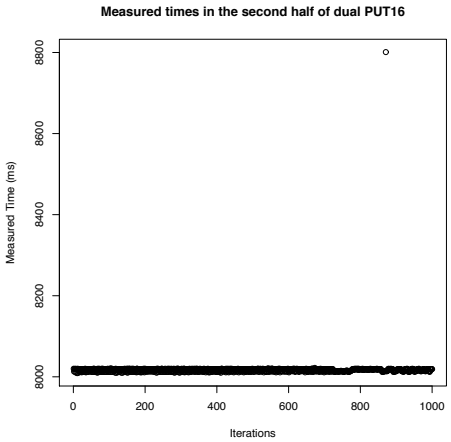
(a) 1st half's measured times on dual PUT8



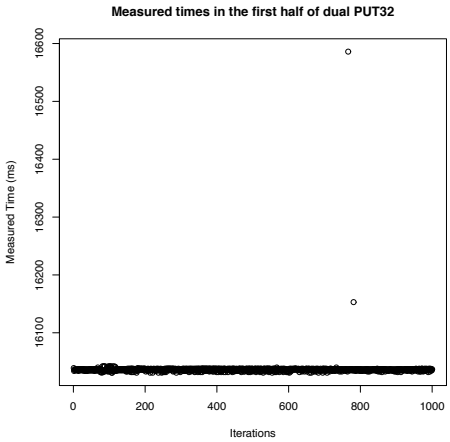
(b) 2nd half's measured times on dual PUT8



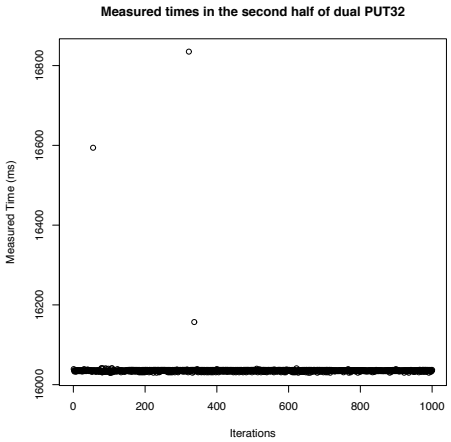
(c) 1st half's measured times on dual PUT16



(d) 2nd half's measured times on dual PUT16

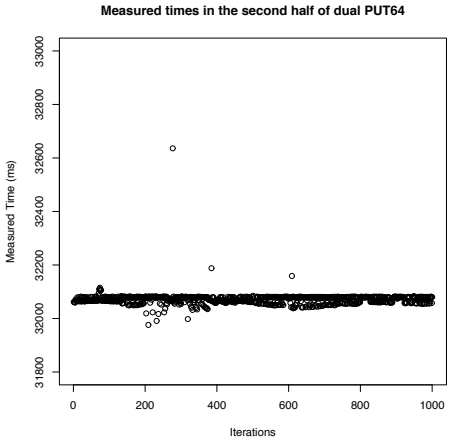


(e) 1st half's measured times on dual PUT32

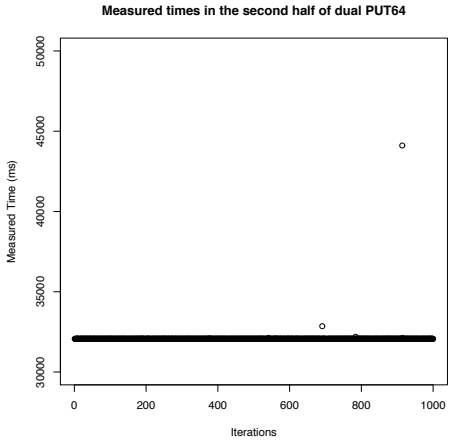


(f) 2nd half's measured times on dual PUT32

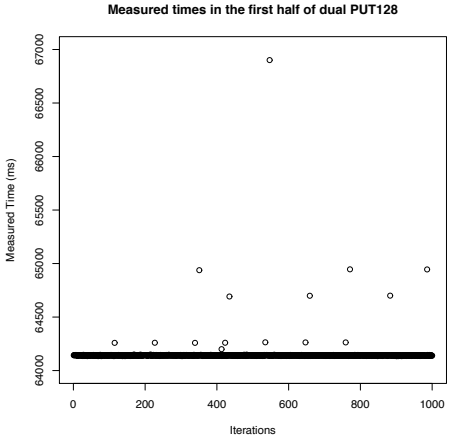
Figure 37: Comparison of measured times on dual PUT8∼PUT32



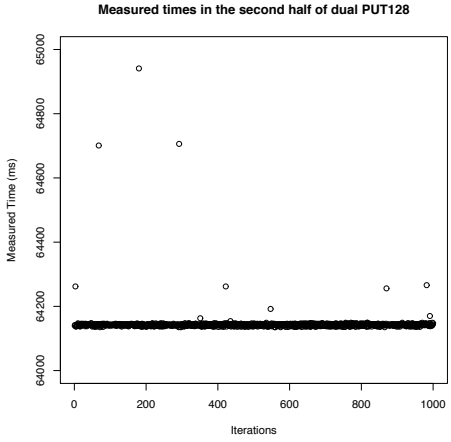
(a) 1st half's measured times on dual PUT64



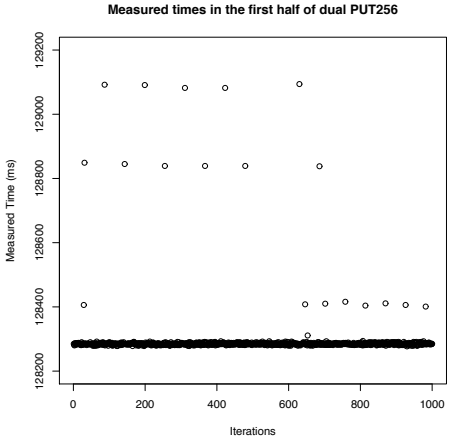
(b) 2nd half's measured times on dual PUT64



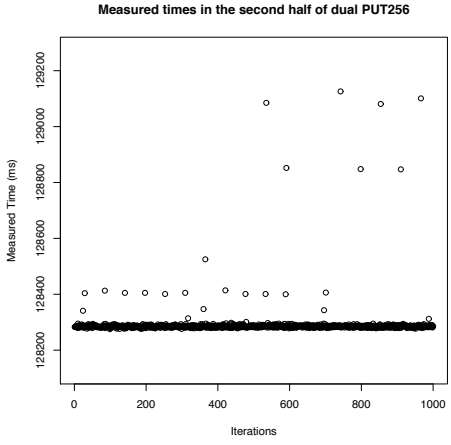
(c) 1st half's measured times on dual PUT128



(d) 2nd half's measured times on dual PUT128

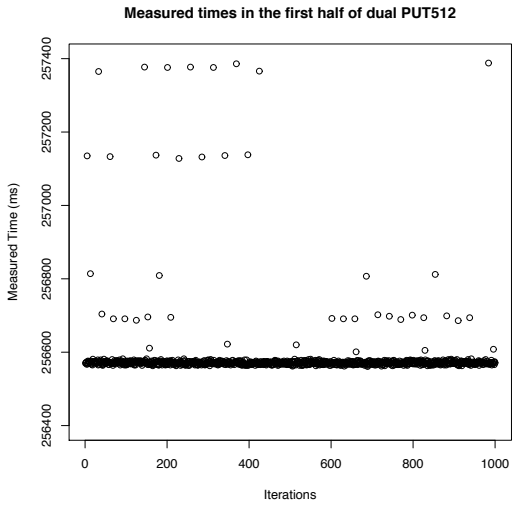


(e) 1st half's measured times on dual PUT256

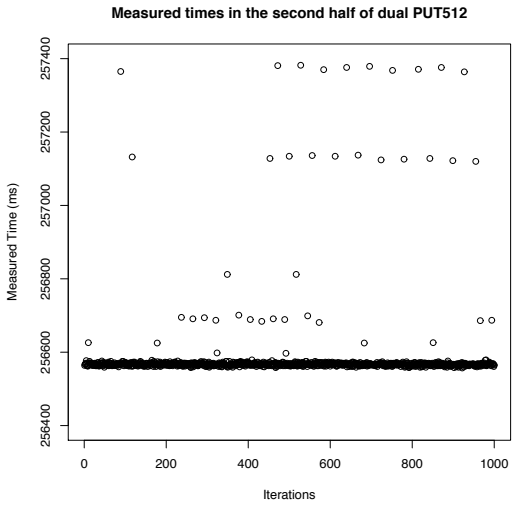


(f) 2nd half's measured times on dual PUT256

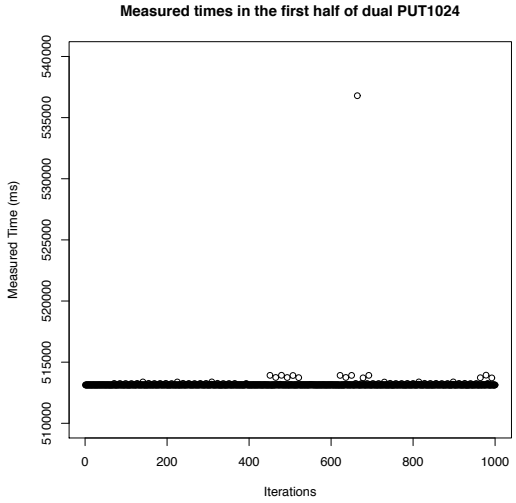
Figure 38: Comparison of measured times onl dual PUT64∼ PUT256



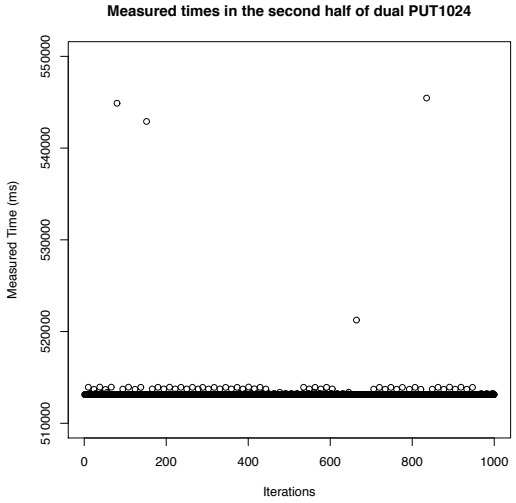
(a) 1st half's measured times on dual PUT512



(b) 2nd half's measured times on dual PUT512

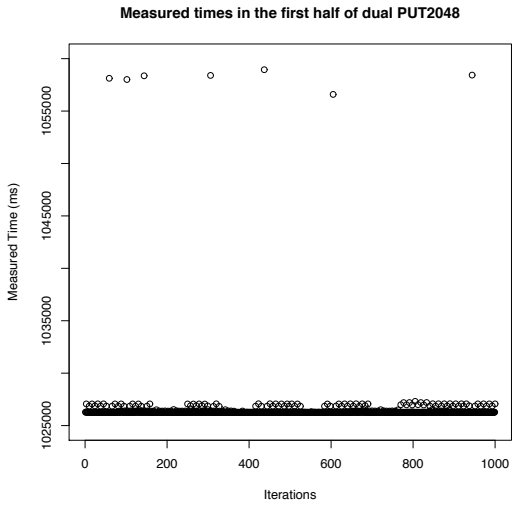


(c) 1st half's measured times on dual PUT1024

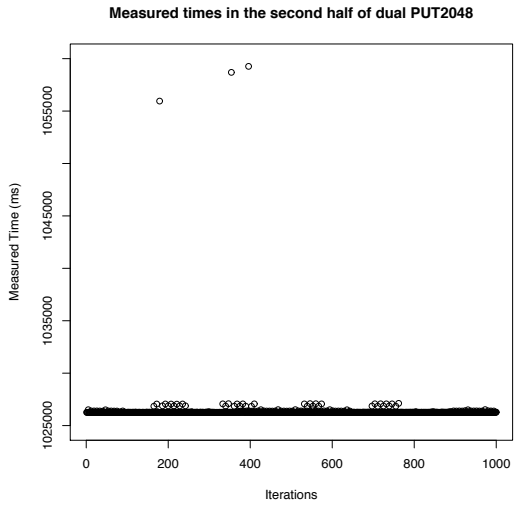


(d) 2nd half's measured times on dual PUT1024

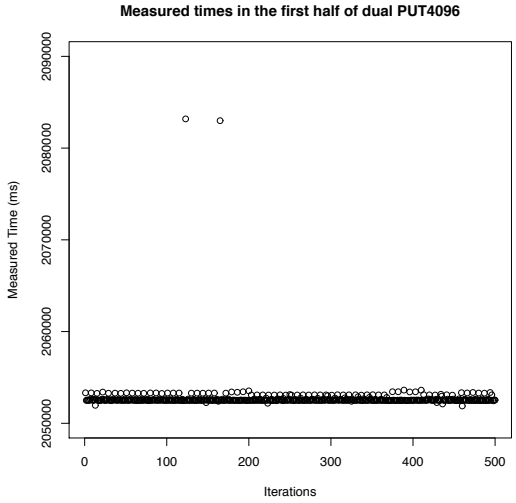
Figure 39: Comparison of measured times on dual PUT512 and PUT1024



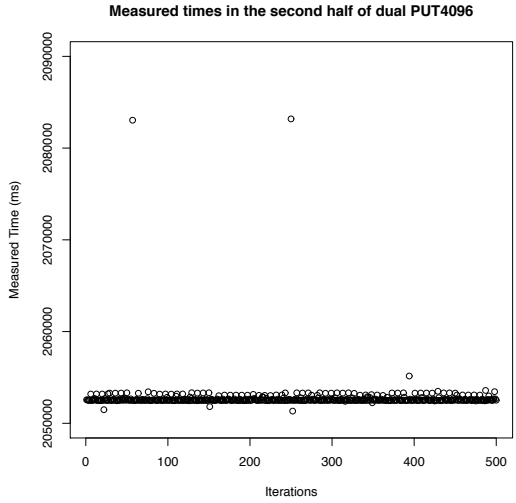
(a) 1st half's measured times on dual PUT2048



(b) 2nd half's measured times on dual PUT2048



(c) 1st half's measured times on dual PUT4096

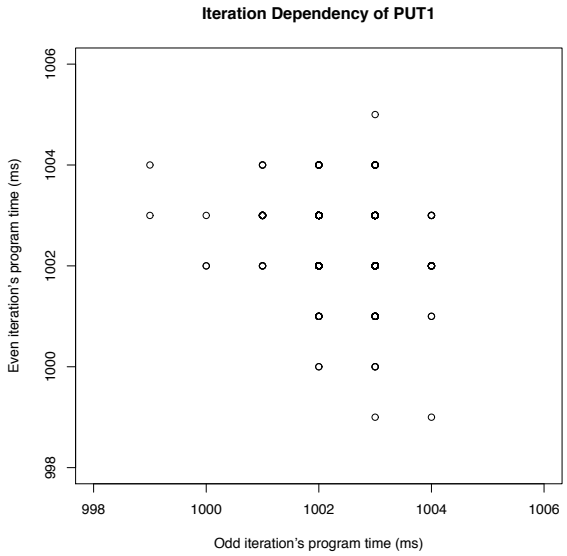


(d) 2nd half's measured times on dual PUT4096

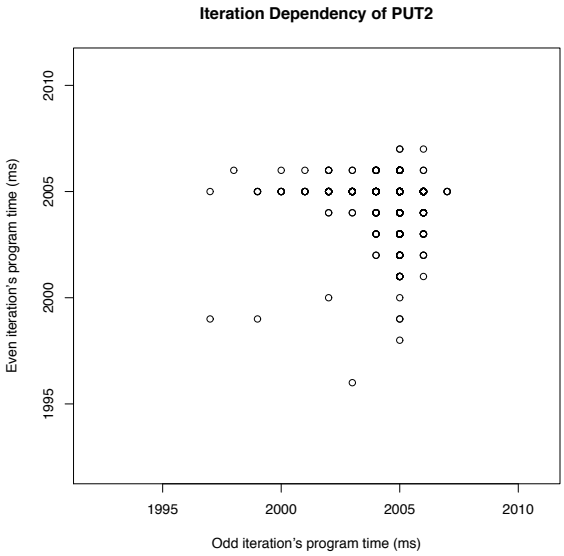
Figure 40: Comparison of measured times on dual PUT2048 and PUT4096

7 Successive Iterations' Dependency

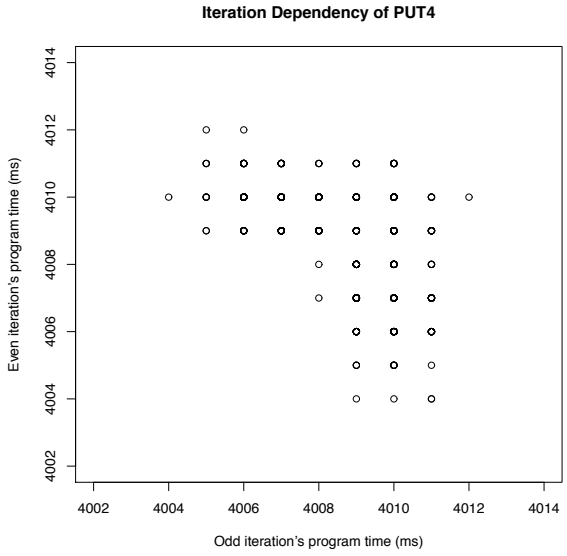
In this section we plot measured times of each iteration pair consisting of odd and even iterations. Specifically, the measured times at adjacent, odd and even iterations consist of x and y coordinates and plotted. The data are described in Table 1. The data in Figure 43 exclude outliers chosen by eye.



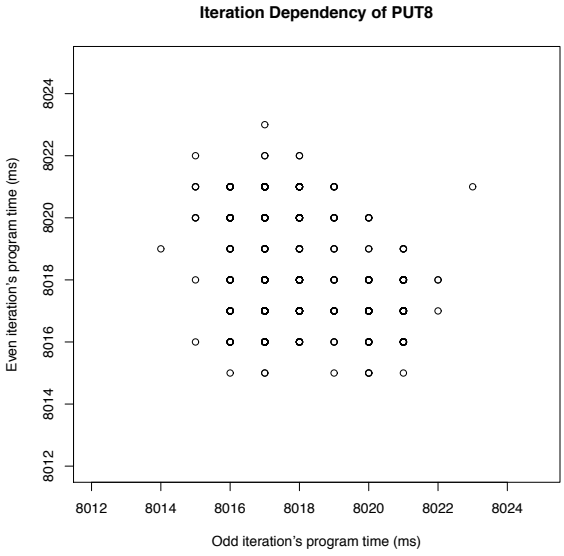
(a) Scatter plot on PUT1



(b) Scatter plot on PUT2

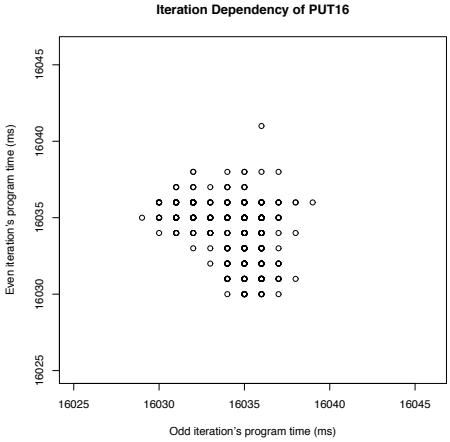


(c) Scatter plot on PUT4

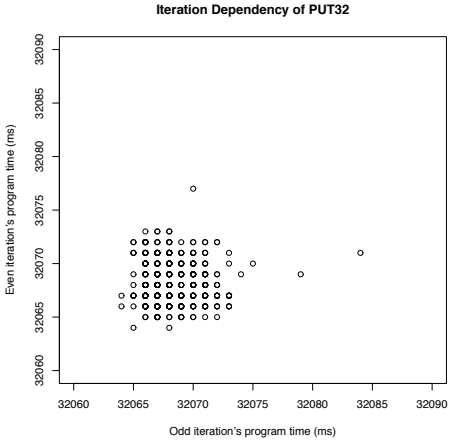


(d) Scatter plot on PUT8

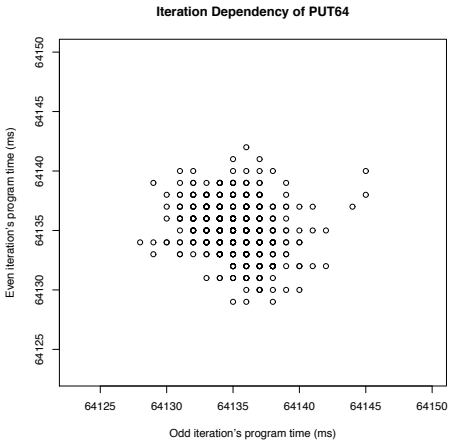
Figure 41: Iteration dependency on PUT1∼PUT8



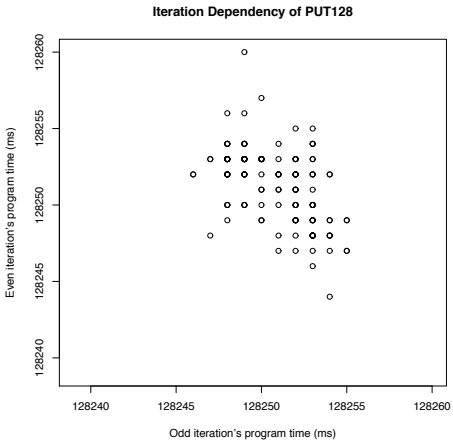
(a) Scatter plot on PUT16



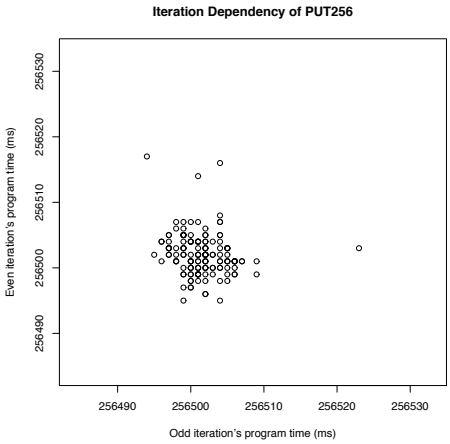
(b) Scatter plot on PUT32



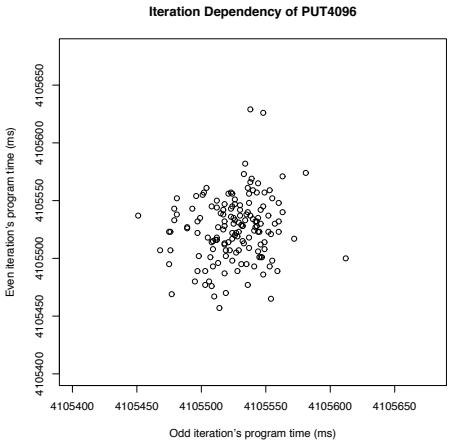
(c) Scatter plot on PUT64



(d) Scatter plot on PUT128

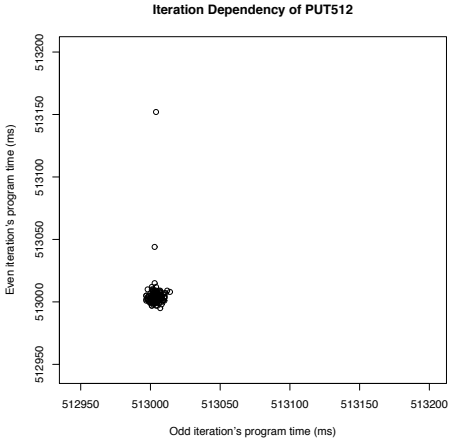


(e) Scatter plot on PUT256

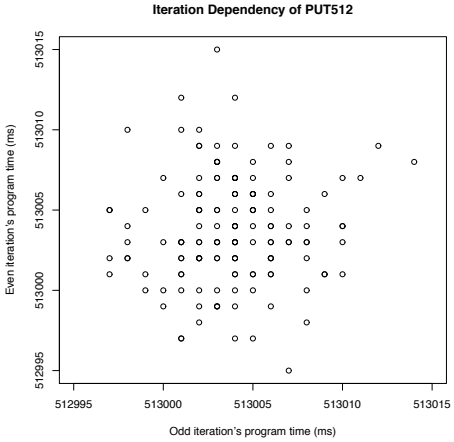


(f) Scatter plot on PUT4096

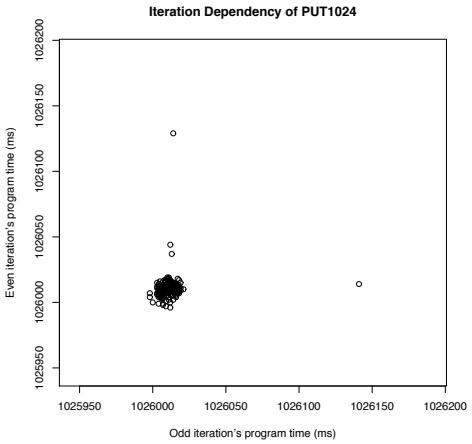
Figure 42: Iteration Dependency on PUT16∼PUT256 and PUT4096



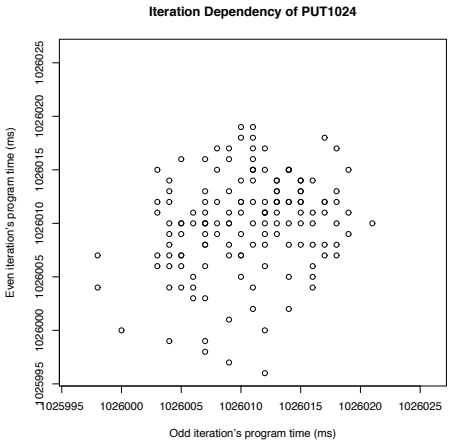
(a) Scatter plot on PUT512



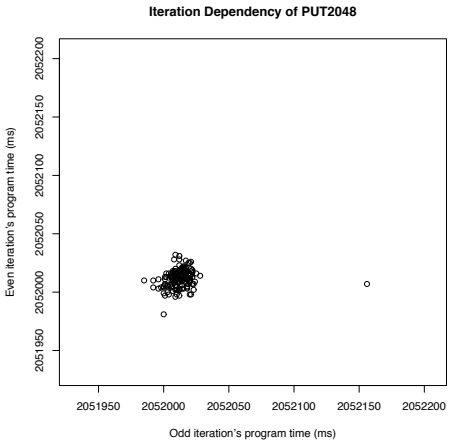
(b) Scatter plot on trimmed data of PUT512



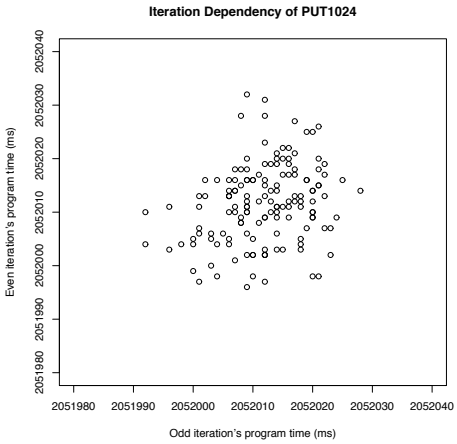
(c) Scatter plot on PUT1024



(d) Scatter plot on trimmed data of PUT1024

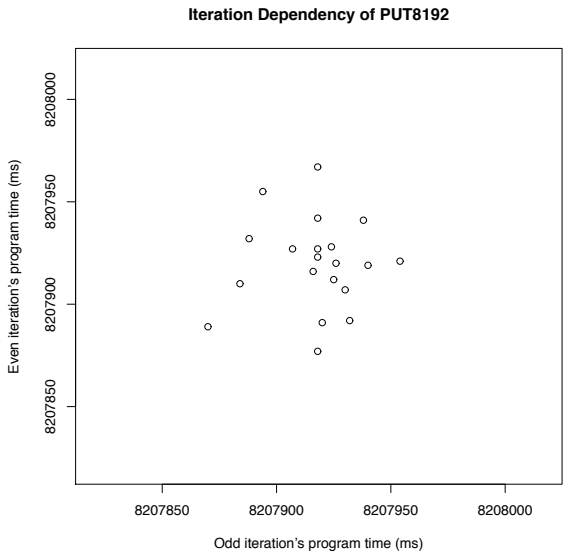


(e) Scatter plot on PUT2048

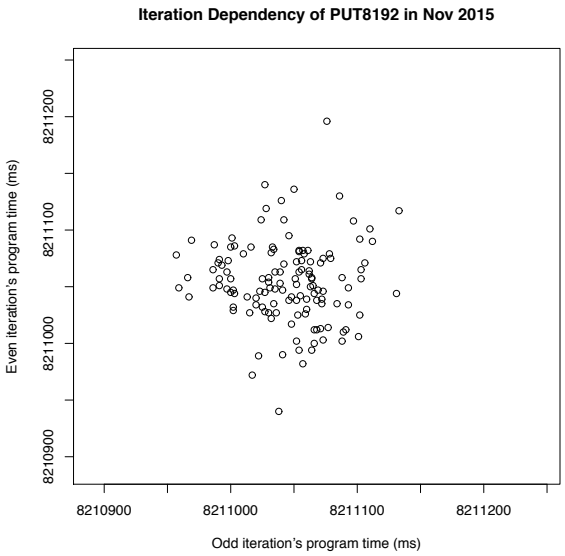


(f) Scatter plot on trimmed data of PUT2048

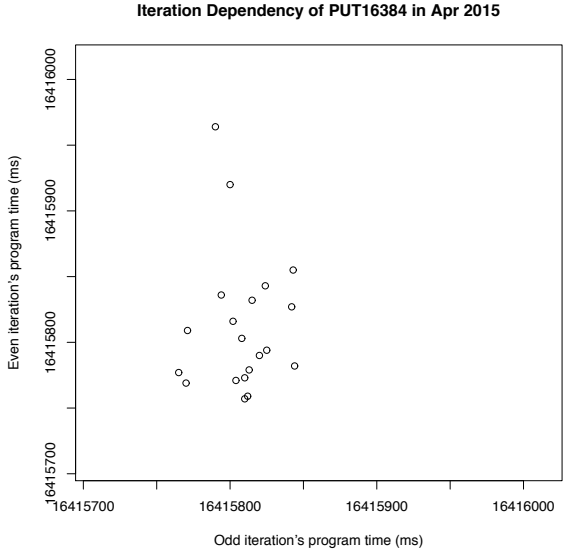
Figure 43: Iteration dependency on PUT512, PUT1024, and PUT2048



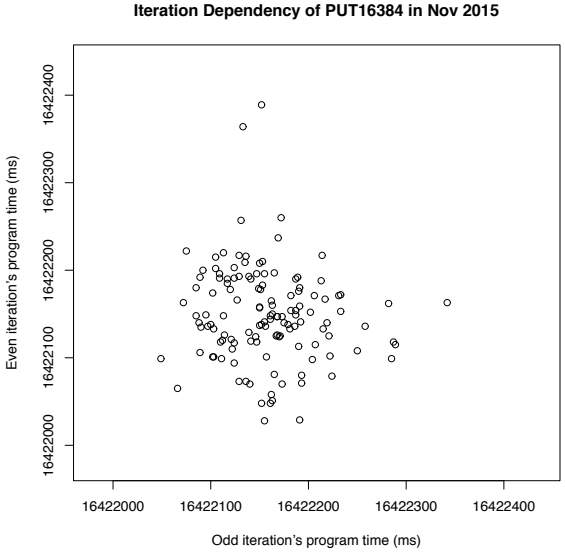
(a) Scatter plot on PUT8192 in Apr 2015



(b) Scatter plot on PUT8192 in Nov 2015



(c) Scatter plot on PUT16384 in Apr 2015



(d) Scatter plot on PUT16384 in Nov 2015

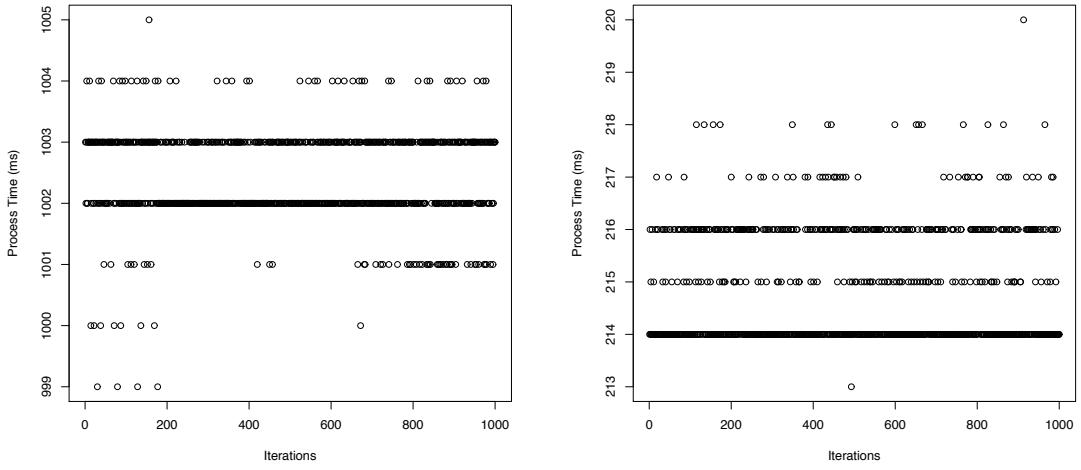
Figure 44: Iteration dependency on PUT8192∼PUT16384

8 Influence of Daemon Process on Program Time Measurement

In this section we investigate correlations of program times between PUT and a group of daemon processes. The base data, obtained by EMPv4, are from Table 1. It seems that the longer PUT, the stronger correlation of its PT with that of daemon processes.

PUT	Correlation Coefficient by EMPv4	Correlation Coefficient by EMPv5-relaxed
PUT1	-0.2	-0.2
PUT2	-0.005	-0.009
PUT4	-0.05	-0.05
PUT8	0.1	0.1
PUT16	0.1	0.1
PUT32	0.3	0.15
PUT64	0.2	0.2
PUT128	0.2	0.2
PUT256	0.4	0.4
PUT512	0.9	0.6
PUT1024	0.9	0.2
PUT2048	0.8	0.24
PUT4096	0.4	0.4
PUT8192 in Apr	0.4	0.4
PUT8192 in Nov	0.3	0.3
PUT16384 in Apr	0.4	0.8
PUT16384 in Nov	0.5	0.5

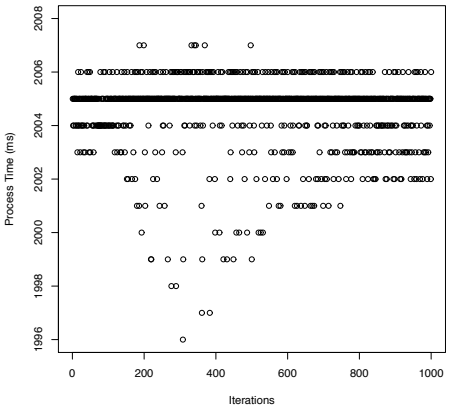
Table 16: Correlation Coefficients between Program Times of Daemon and PUT by EMPv4 and EMPv5



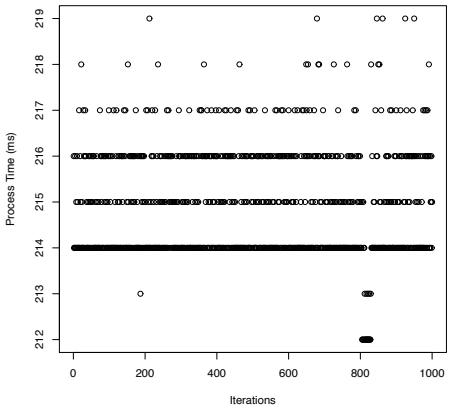
(a) PUT1's PTs

(b) PUT1's daemon PTs

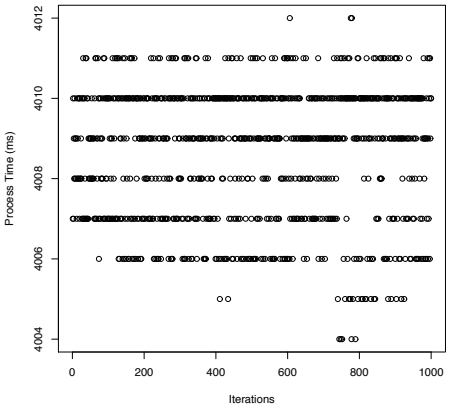
Figure 45: Program times between PUT1 vs. Daemon processes



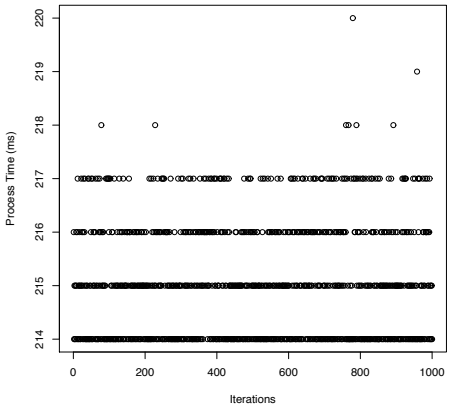
(a) PUT2's PTs



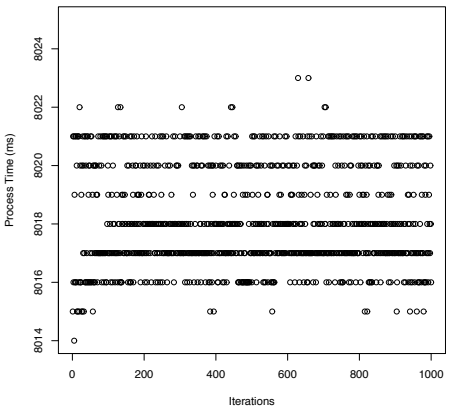
(b) PUT2's daemon PTs



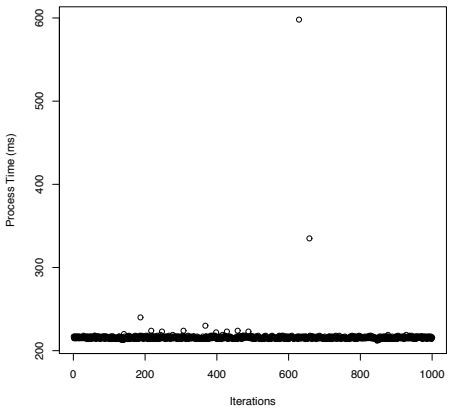
(c) PUT4's PTs



(d) PUT4's daemon PTs

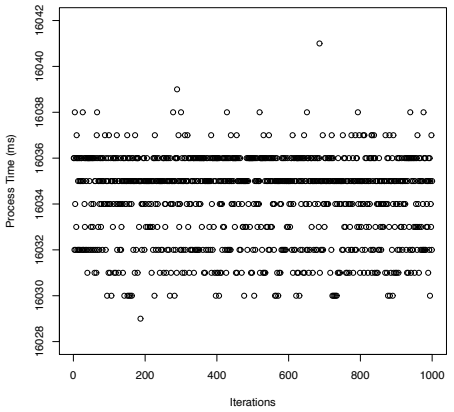


(e) PUT8's PTs

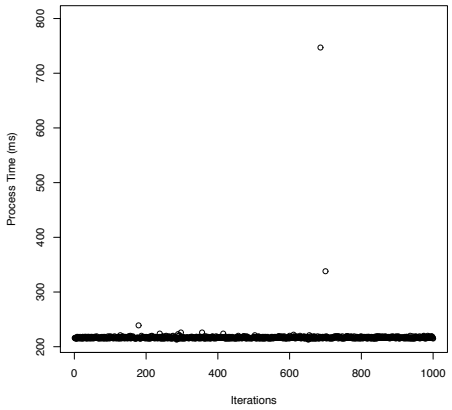


(f) PUT8's daemon PTs

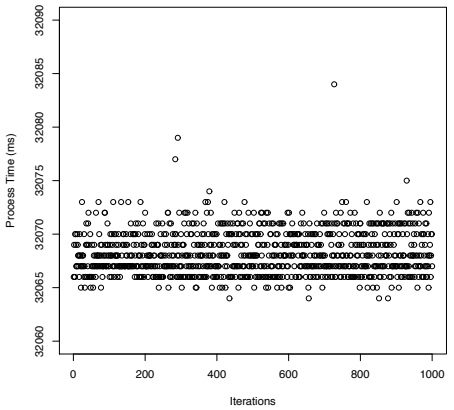
Figure 46: Program times between PUT2∼PUT8 vs. Daemon processes



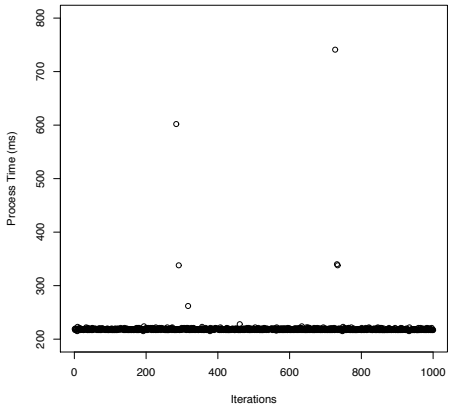
(a) PUT16's PTs



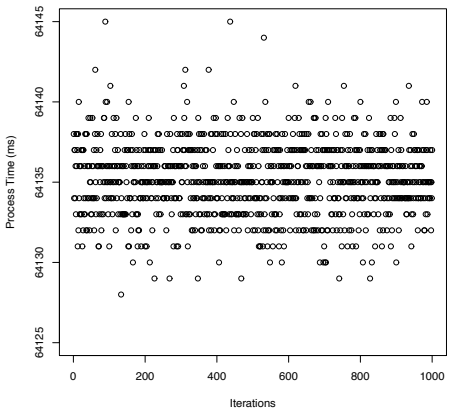
(b) PUT16's daemon PTs



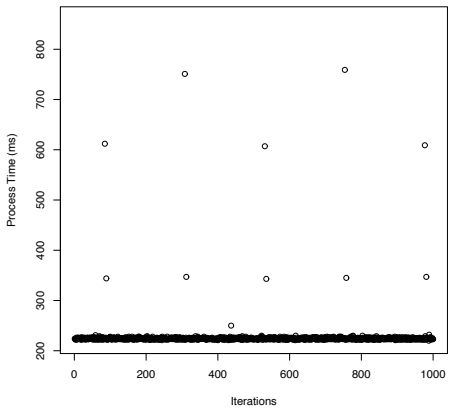
(c) PUT32's PTs



(d) PUT32's daemon PTs

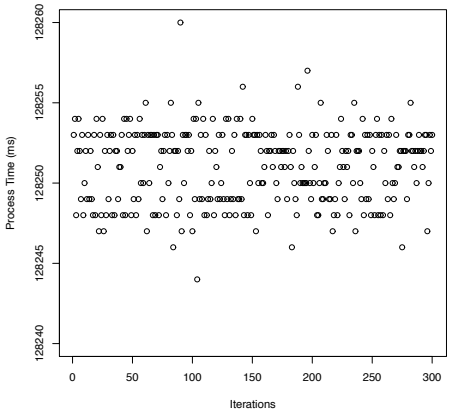


(e) PUT64's PTs

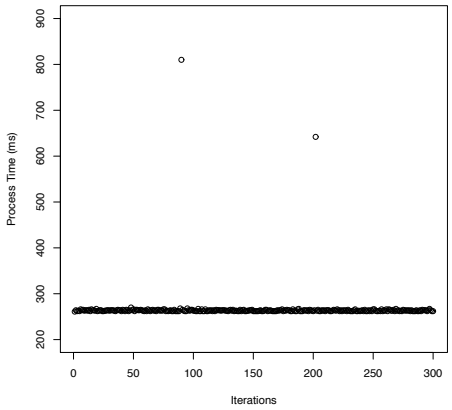


(f) PUT64's daemon PTs

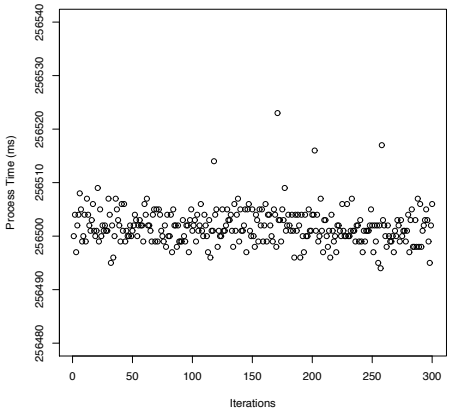
Figure 47: Program times between PUT16∼PUT64 vs. Daemon processes



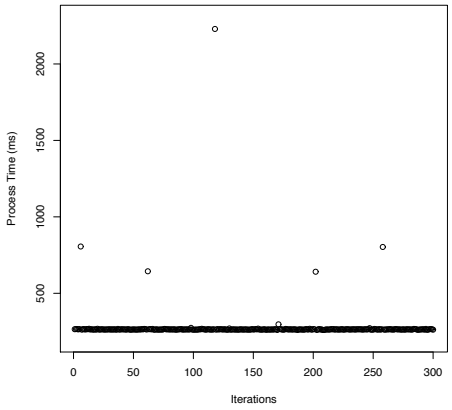
(a) PUT128's PTs



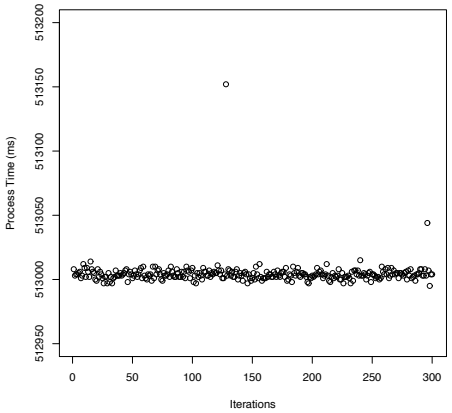
(b) PUT128's daemon PTs



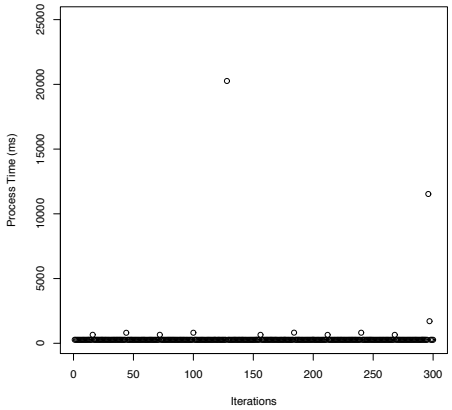
(c) PUT256's PTs



(d) PUT256's daemon PTs

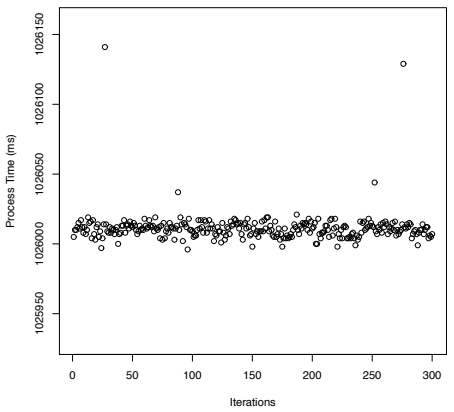


(e) PUT512's PTs

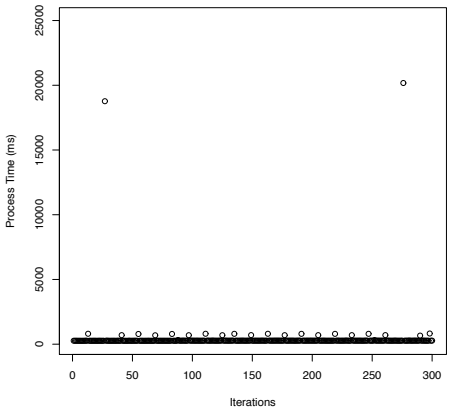


(f) PUT512's daemon PTs

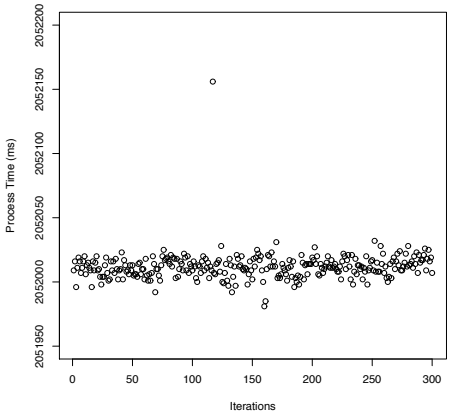
Figure 48: Program times between PUT128∼PUT512 vs. Daemon processes



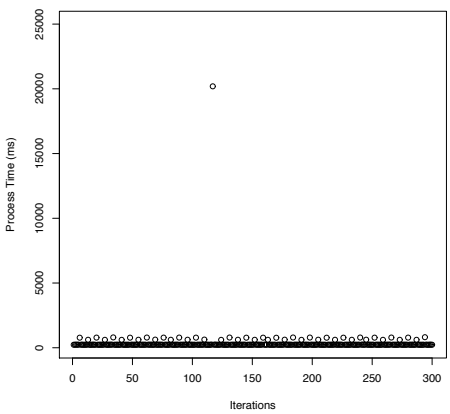
(a) PUT1024's PTs



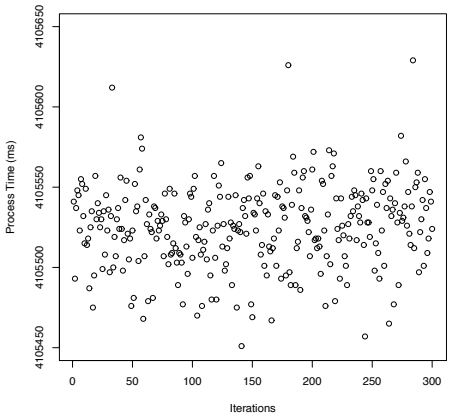
(b) PUT1024's daemon PTs



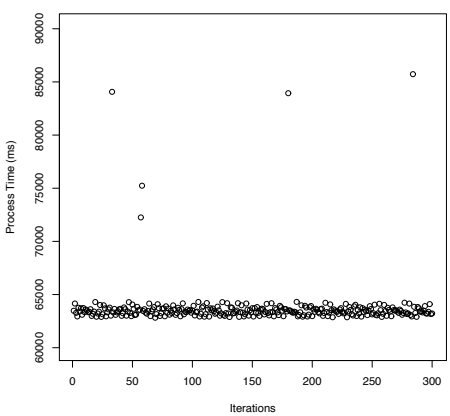
(c) PUT2048's PTs



(d) PUT2048's daemon PTs

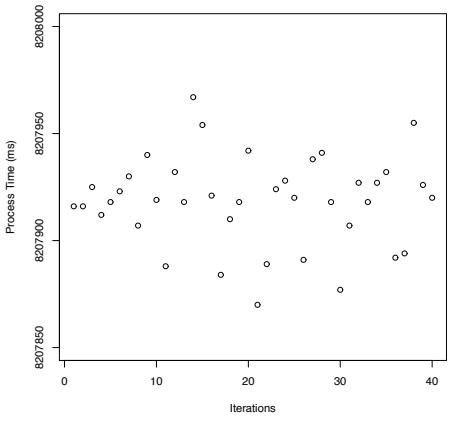


(e) PUT4096's PTs

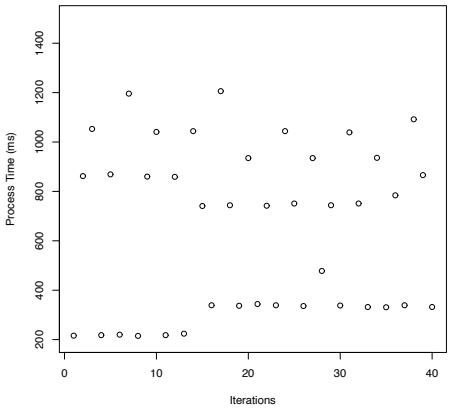


(f) PUT4096's daemon PTs

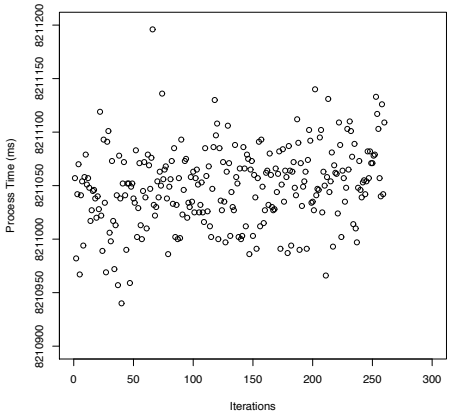
Figure 49: Program times between PUT1024∼PUT4096 vs. Daemon processes



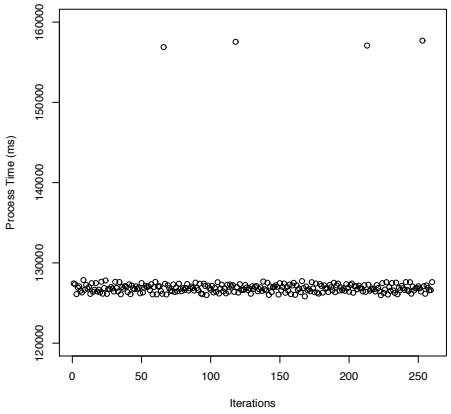
(a) PUT8192's PTs in Apr 2015



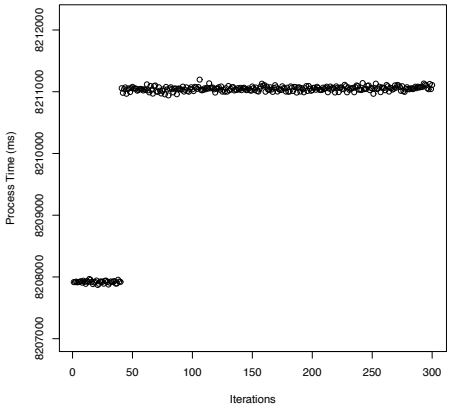
(b) PUT8192's daemon PTs in Apr 2015



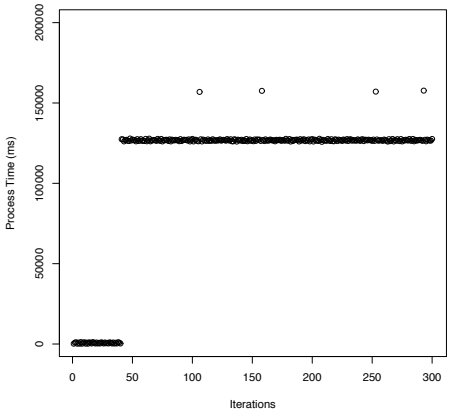
(c) PUT8192's PTs in Nov 2015



(d) PUT8192's daemon PTs in Nov 2015

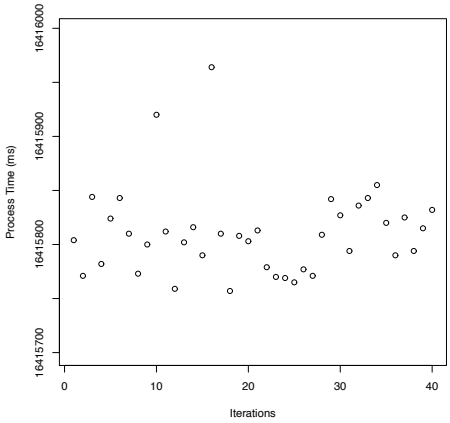


(e) Combined PUT8192's PTs

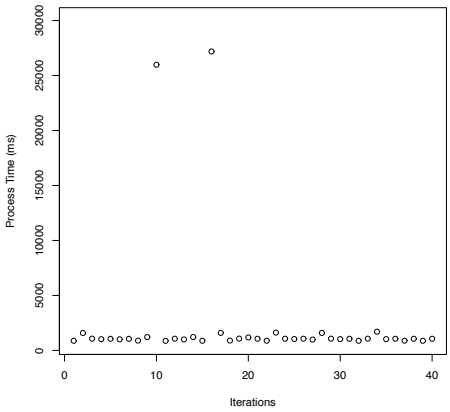


(f) Combined PUT8192's daemon PTs

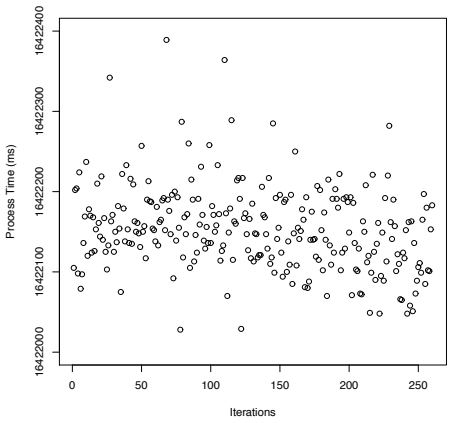
Figure 50: Program times between PUT8192 vs. Daemon processes



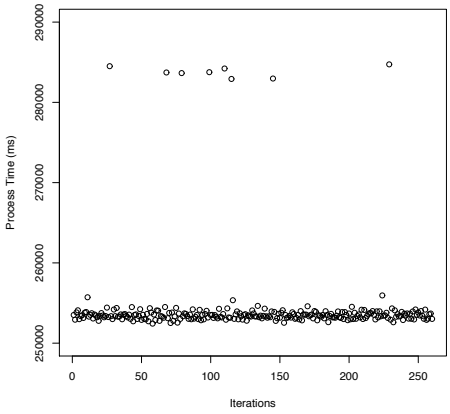
(a) PUT16384's PTs in Apr 2015



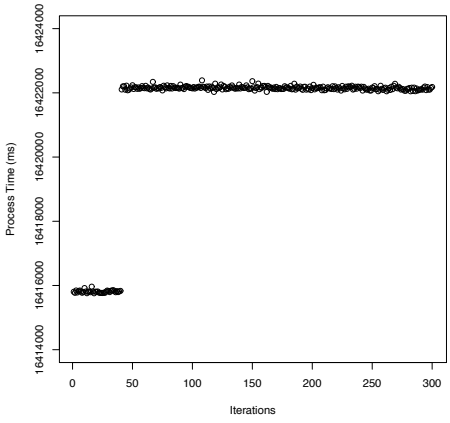
(b) PUT16384's daemon PTs in Apr 2015



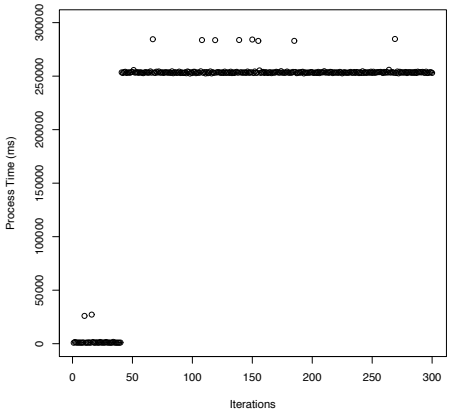
(c) PUT16384's PTs in Nov 2015



(d) PUT16384's daemon PTs in Nov 2015



(e) Combined PUT16384's PTs



(f) Combined PUT16384's daemon PTs

Figure 51: Program times between PUT16384 vs. Daemon processes

9 Conclusion

Below are the summarized observations by this study. (Order does not matter.)

- Typically, the distribution of program times (PTs) of PUT is somewhat mixture of two (or possibly more) models. (But only when you compare fist/second half.)
- Outlier trimming does not well shape a normal distribution of PT. Outliers are higher non-normal. Still, we don't have a precise characterization of outliers, but doing so would be difficult because of very large, very infrequent outliers (cf. Figure 12).
- For a short task length of PUT, it seems there is dependency between iterations in the same run of PUT.
- Presence and activation of (infrequent) daemon processes strongly contribute to creating high variance in PT measurement, due to a few very-far outliers. This results in a very long, very thin tail with a structure we don't understand.
- The bigger task length, the stronger correlation between PTs of PUT and daemon processes, probably because the longer the PUT runs, the more likely that there will be a daemon process. **Caveat**: what if the task size is not large and the sample size is small?
- **New:** The bigger task length, the more clumps formed, probably caused by different periodicity of different daemon processes (cf. Figures 23 and 24). This implies that when a given task length is big, it will give more room to capture activities of various daemon processes at different times. Accordingly, it leads to observing mixture of two (or more) models, as addressed in the second item.
- When PUT is timed affects PT distribution, due to the presence of daemon processes whose running can't be controlled.
- Measurement protocol is scalable with growing sample size and increasing task length, because the protocol can do the trimming: see above. **Idea**: Run one version *many* times to estimate the distribution and what is a reasonable criterion for dropping outliers.

Below is a remaining issue(s) revealed through this study.

- We want a definition of "outlier" over a wide range of program times.

10 Appendix

This appendix provides specific details of what daemon processes was captured and how much time was taken at a specific iteration revealing the most program time of a certain PUT.

10.1 Breakdown on Program Times of Daemon Processes

PUT256	Program Time
`incr_work`	256,514 msecs (at the 118th iteration)
Daemon Processes	Program Time
`java`	2 msecs
`md0_raid1`	4 msecs
`jbd2/md0-8`	1 msec
`flush-9:0`	10 msecs
`proc_monitor`	262 msecs
`rhnsd`	6 msecs
`rhn_check`	1,944 msecs
Total	2,229 msecs

Table 17: The daemon processes captured at the worst PT of PUT256

PUT256	Program Time
`incr_work`	513,152 msecs (at the 128th iteration)
Daemon Processes	Program Time
`java`	2 msecs
`md0_raid1`	51 msecs
`jbd2/md0-8`	27 msecs
`flush-9:0`	86 msecs
`proc_monitor`	270 msecs
`rhnsd`	6 msecs
`rhn_check`	19,820 msecs
Total	20,262 msecs

Table 18: The daemon processes captured at the worst PT of PUT512

PUT4096	Program Time
incr_work	4,105,629 msecs (at the 284th iteration)
Daemon Processes	Program Time
events/0	1 msec
kblockd/0	1 msec
kslowd000	31,710 msecs
kslowd001	31,782 msecs
md0_raid1	82 msecs
jbd2/md0-8	21 msecs
flush-9:0	79 msecs
proc_monitor	206 msecs
rhnsd	3 msecs
ntpd	1 msec
java	2 msecs
rhn_check	21,840 msecs
Total	85,728 msecs

Table 19: The daemon processes captured at the worst PT of PUT4096

PUT8192	Program Time
incr_work	8,207,884 msecs (at the 244th iteration)
Daemon Processes	Program Time
kblockd/0	3 msecs
kslowd000	31,710 msecs
kslowd001	31,782 msecs
md0_raid1	12 msecs
jbd2/md0-8	2 msecs
proc_monitor	204 msecs
rhnsd	6 msecs
java	1 msec
rhsmcertd-worke	114 msecs
rhsmcertd-worke	114 msecs
rhn_check	708 msecs
Total	64,656 msecs

Table 20: The daemon processes captured at the worst PT of PUT8192

Daemon Processes	Descriptions
`kslowd000` (`kslowd001`)	A kernel threads for performing things that take a relatively long time. "Typically, when processing something, these items will spend a lot of time, blocking a thread on I/O, thus making that thread unavailable for doing other work." (`http://www.mjmwired.net/kernel/Documentation/slow-work.txt`)
`rhn_check`	An external program for check for updates, run by `rhnsd`
`rhnsd`	"A background daemon process that periodically polls the Red Hat Network to see if there are any queued actions available. Typically started from the initialization (`init`) scripts in `/etc/init.d/rhnsd` when its time to poll the Red Hat Network server for available updates and actions. The default interval is every 240 minutes. The minimum polling interval is 60 minutes. Any network activity is done via the `rhn_check` utility." (`http://linuxcommand.org/man_pages/rhnsd8.html`)

Table 21: Descriptions of some daemon processes