

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования “Национальный исследовательский
университет ИТМО”

**ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И
КОМПЬЮТЕРНОЙ ТЕХНИКИ**

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине

“Операционные системы”

выполнил:

Иванов Матвей Сергеевич

группа Р33111

Преподаватель:

Пашнин Александр Денисович

Осипов Святослав Владимирович

г. Санкт-Петербург, 2023

Оглавление

Оглавление	2
Задание	3
Ход работы	4
Анализ производительности CPU	4
Анализ производительности cache	10
Анализ производительности IO	14
Анализ производительности memory	18
Анализ производительности network	20
Анализ производительности pipe	23
Анализ производительности sched	26
Заключение	30

Задание

Основная цель лабораторной работы — знакомство с системными инструментами анализа производительности и поведения программ. В данной лабораторной работе Вам будет предложено произвести нагружочное тестирование Вашей операционной системы при помощи инструмента stress-ng.

В качестве тестируемых подсистем использовать: cri, cache, io, memory, network, pipe, scheduler.

Для работы со счетчиками ядра использовать все утилиты, которые были рассмотрены на лекции (раздел 1.9, кроме kdb)

Ниже приведены списки параметров для различных подсистем (Вам будет выдано 2 значения для каждой подсистемы согласно варианту в журнале). Подбирая числовые значения для выданных параметров, и используя средства мониторинга, добиться максимальной производительности системы (BOGOPS, FLOPS, Read/Write Speed, Network Speed).

Параметры для cri:

gcd hyperbolic

Параметры для cache:

cache-level l1cache-ways

Параметры для io:

iomix ioprio

Параметры для memory:

mcontend shm

Параметры для network:

netlink-proc netlink-task

Параметры для pipe:

pipeherd-yield sigpipe

Параметры для sched:

sched-period schedpolicy

Построить графики (подходящие по заданию.):

- Потребления программой CPU;
- Нагрузки, генерируемой программой на подсистему ввода-вывода;
- Нагрузки, генерируемой программой на сетевую подсистему;
- Другие графики, необходимые для демонстрации работы.

Ход работы

Анализ производительности CPU

В данном разделе мы рассмотрим параметры `gcd` и `hyperbolic` для CPU. Чтобы проанализировать производительность будем отслеживать показатель `BOGO OPS`, который нам выдаёт `stress-ng`. Но стоит помнить, что хоть и этот параметр является весьма показательным, он не является достаточно достоверным, основываясь на документации `stress-ng`. Помимо этого показателя, также возьмём во внимание показания снятые с помощью `sar -u` и `htop`.

Ну и так как мы ищем максимальные показатели, а не хотим чтобы наш компьютер сгорел (утилита `stress-ng` вполне может навредить системе), так что выберем разумные границы нашего анализа.

Команды для анализа с изменением количества воркеров:

```
stress-ng --cpu N --cpu-method gcd -t 15s --metrics | sar 1 10 -u  
stress-ng --cpu N --cpu-method hyperbolic -t 15s --metrics | sar 1 10 -u
```

Где `N` – количество воркеров

Напишем скрипт на языке Bash для тестирования:

```
#!/bin/bash  
WORKERS_START=1  
WORKERS_END=6  
TARGET="$1"  
if [ -z $TARGET ]  
  
then  
    echo "Pass target cpu method"  
else  
    echo "Analyze $TARGET"  
    for i in $(seq $WORKERS_START $WORKERS_END)  
    do echo -e "\nstart $i worker(s)\n";  
        stress-ng --cpu $i --cpu-method $TARGET -t 15s --metrics | sar 1 10 -u  
    done  
fi
```

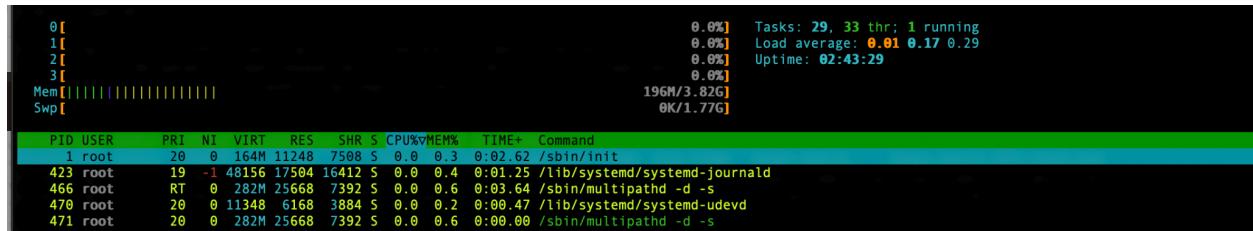
Результаты работы можно увидеть по ссылкам:

Для gcd: <https://pastebin.com/thnMPYPZ>

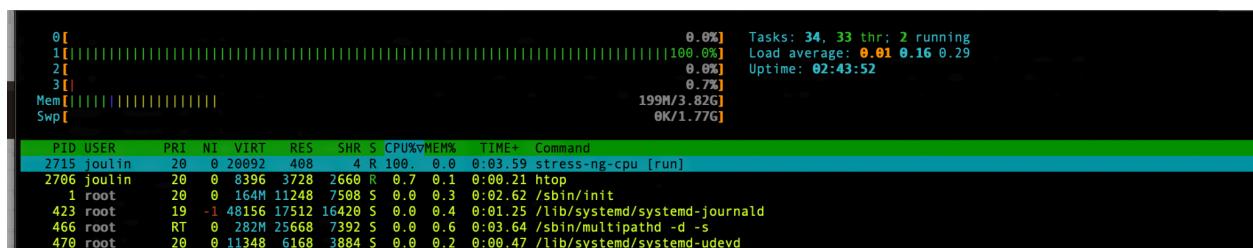
Для hyperbolic: <https://pastebin.com/s6MafQiX>

Скрины htop при работе скрипта для gcd:

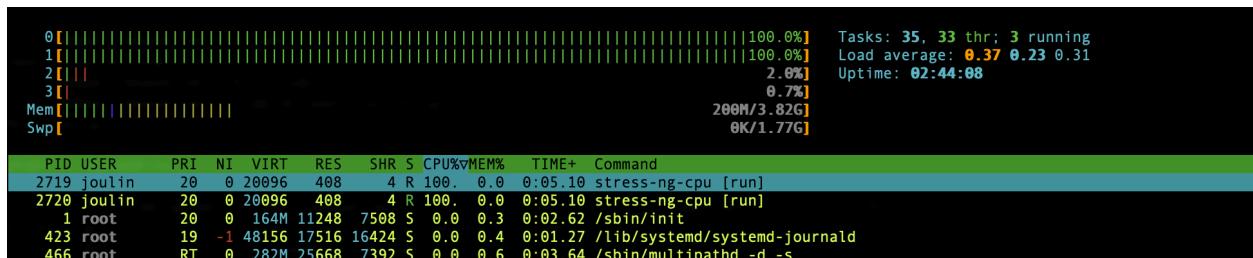
До запуска



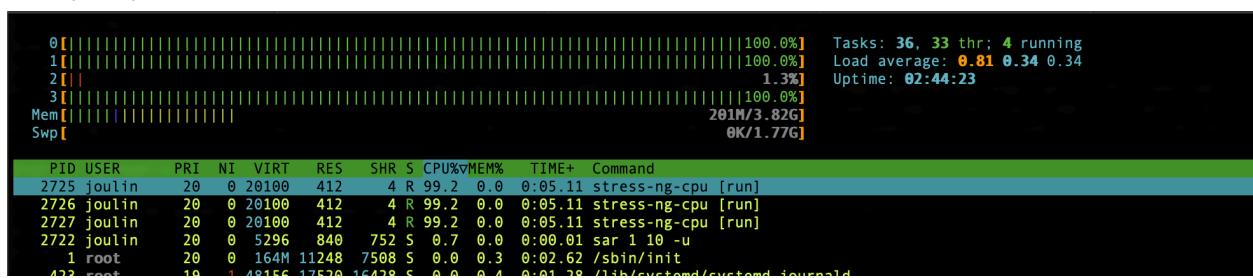
1 воркер



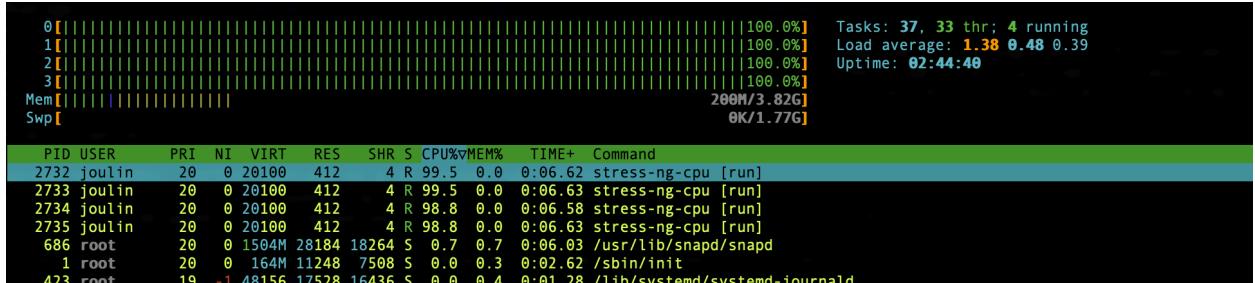
2 воркера



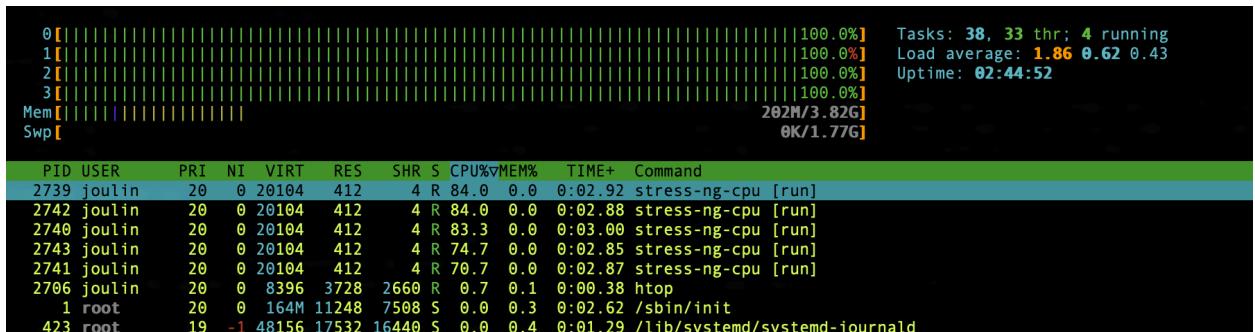
3 воркера



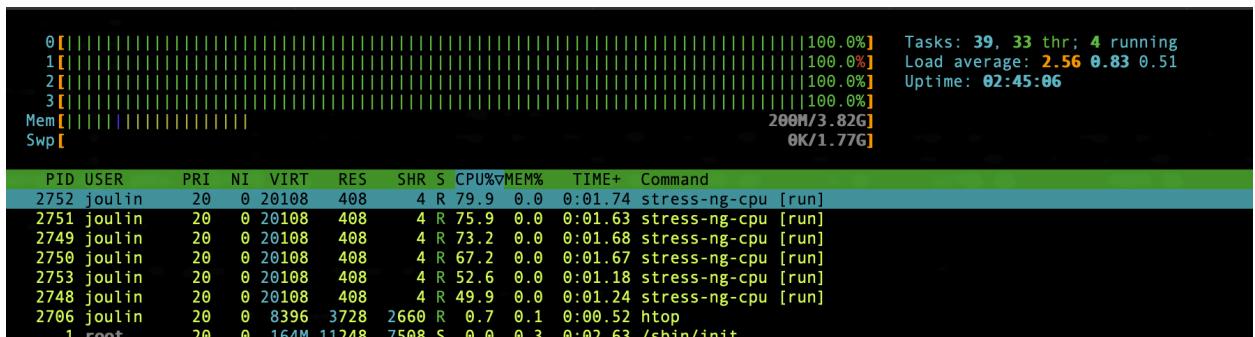
4 воркера



5 воркеров

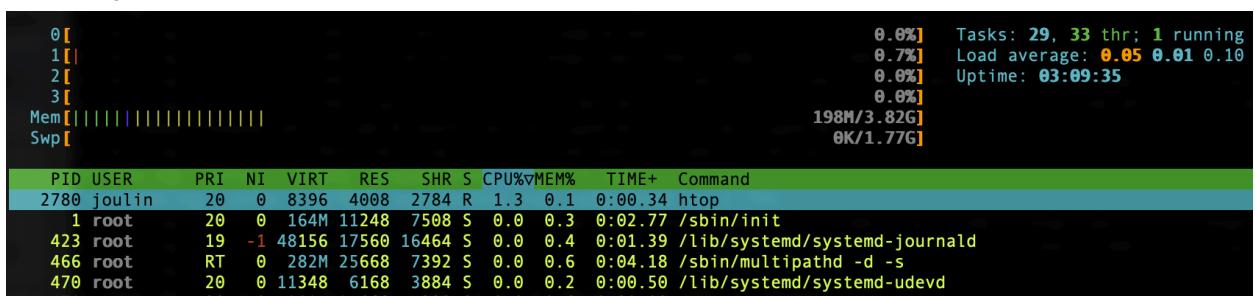


6 воркеров

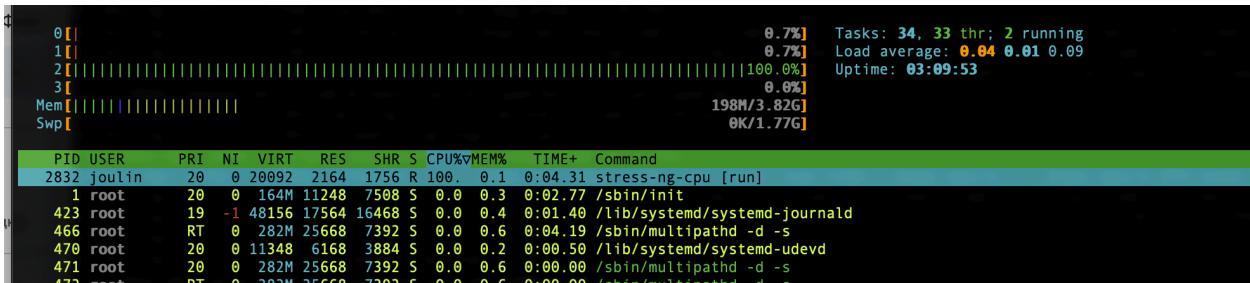


Скринь htop при работе скрипта для hyperbolic:

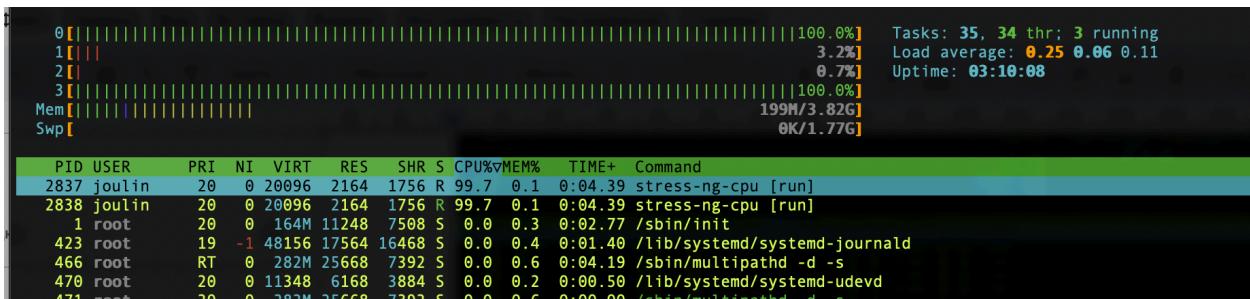
До запуска



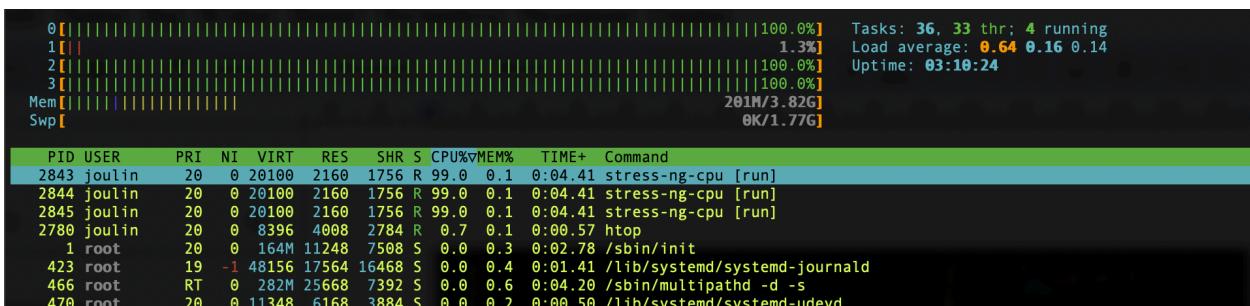
1 воркера



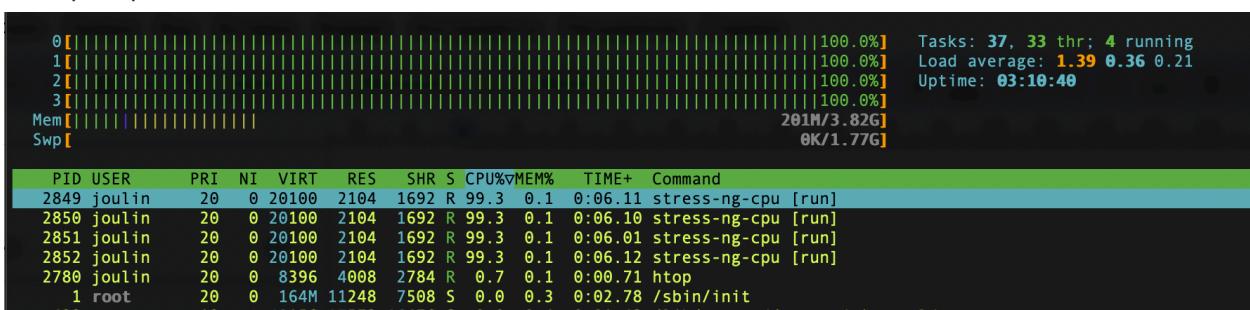
2 воркера



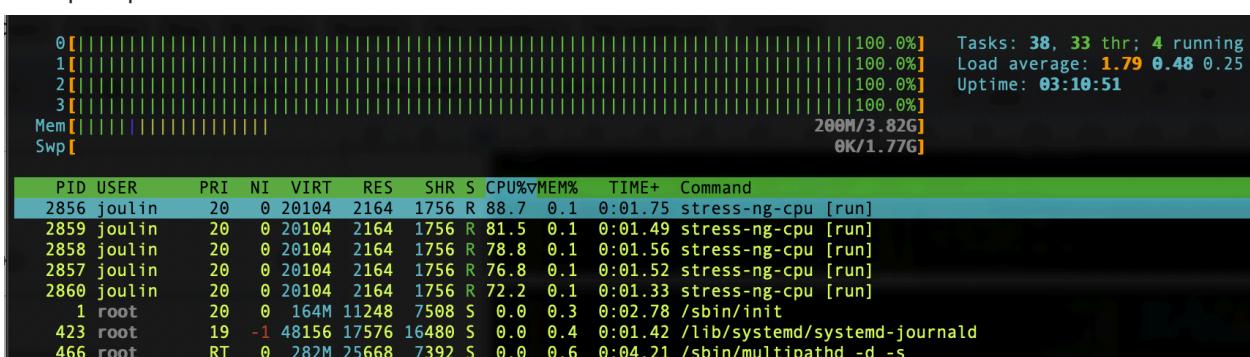
3 воркера



4 воркера



5 воркеров



6 воркеров

```

0[|||||] 1[|||||] 2[|||||] 3[|||||] 100.0% Tasks: 39, 33 thr; 4 running
1[|||||] 100.0% Load average: 2.42 0.68 0.32
2[|||||] Uptime: 03:11:06
3[|||||] 100.0%
Mem[|||||] 201M/3.82G
Swp[|||||] 0K/1.77G

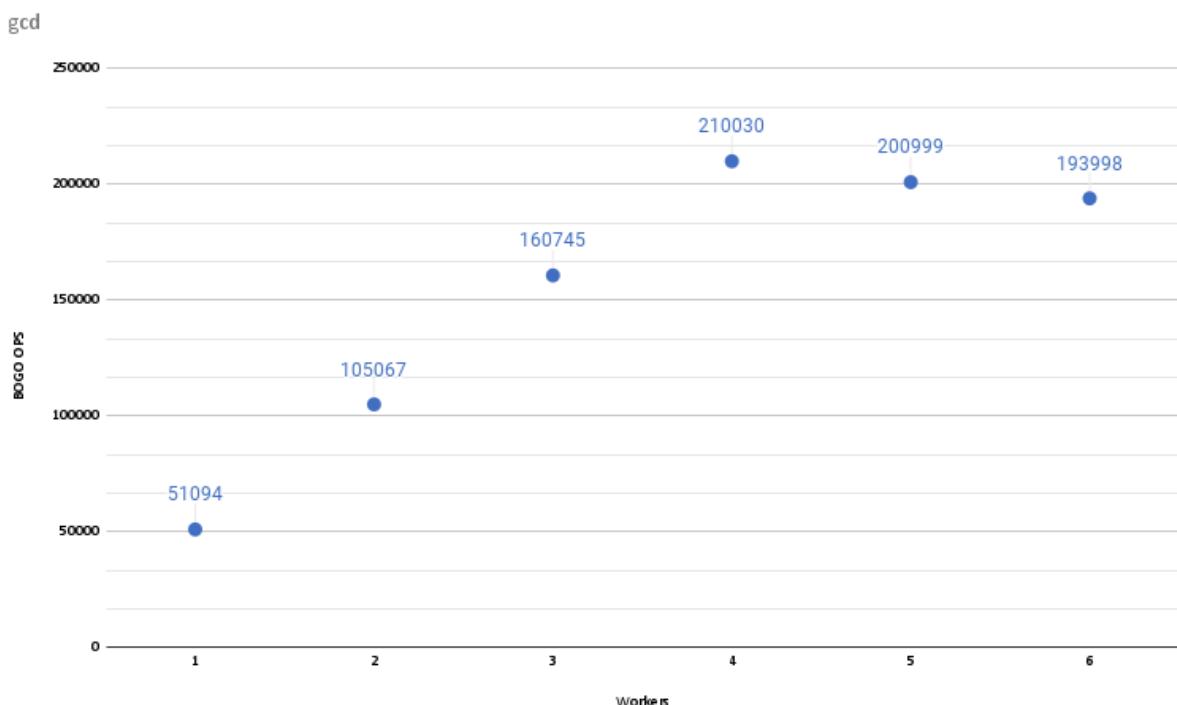
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
2867 joulin 20 0 20108 2096 1692 R 81.5 0.1 0:01.48 stress-ng-cpu [run]
2869 joulin 20 0 20108 2096 1692 R 75.5 0.1 0:01.39 stress-ng-cpu [run]
2865 joulin 20 0 20108 2096 1692 R 71.5 0.1 0:01.56 stress-ng-cpu [run]
2868 joulin 20 0 20108 2096 1692 R 69.5 0.1 0:01.46 stress-ng-cpu [run]
2866 joulin 20 0 20108 2096 1692 R 50.3 0.1 0:01.00 stress-ng-cpu [run]
2864 joulin 20 0 20108 2096 1692 R 49.7 0.1 0:01.05 stress-ng-cpu [run]
2780 joulin 20 0 8396 4008 2784 R 1.3 0.1 0:00.93 htop
1 root 20 0 164M 11248 7588 S 0.0 0.3 0:02.78 /sbin/init

```

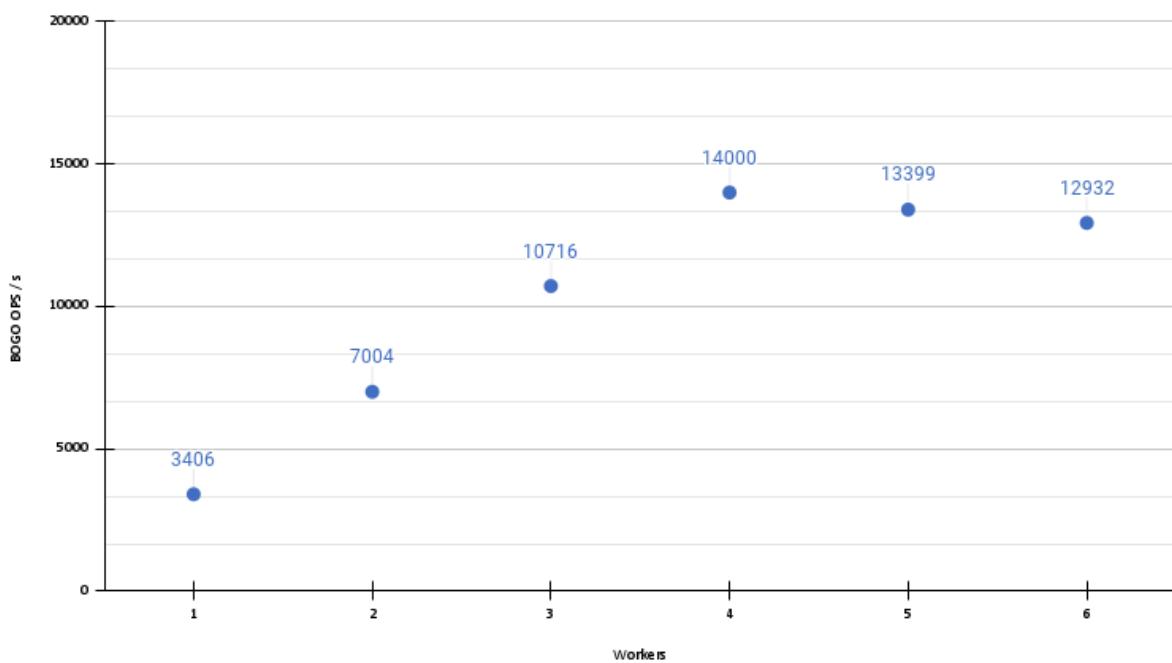
Проанализируем результаты:

CPU						
gcd				hyperbolic		
Workers	bogo ops	bogo ops / s	avg %user	bogo ops	bogo ops / s	avg %user
1	51094	3406	24,89	1746	116	24,96
2	105067	7004	49,59	3456	230	49,7
3	160745	10716	74,56	4710	314	74,44
4	210030	14000	99,92	4645	309	99,62
5	200999	13399	99,87	5361	357	99,87
6	193998	12932	99,95	5412	360	99,75

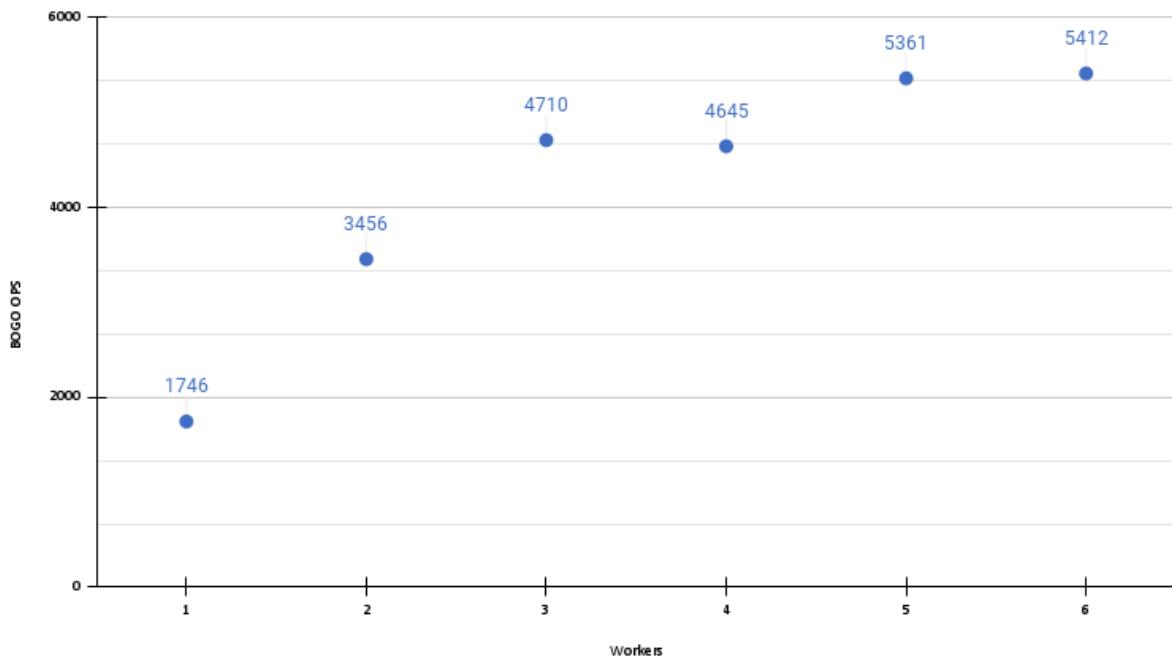
Графики:

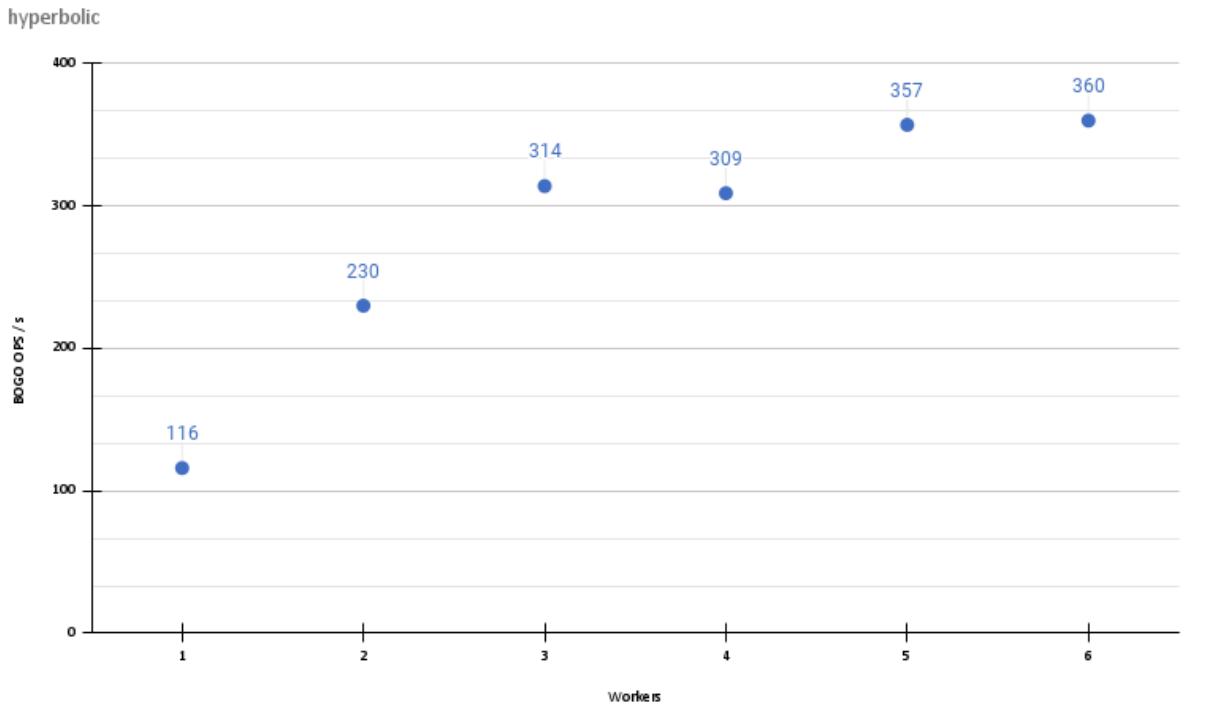


gcd



hyperbolic





Как видно по графикам, при достижении количества воркеров количества ядер, значения перестают сильно изменяться и даже наоборот, постепенно ухудшаются. Но также интересный результат наблюдается на показателе `hyperbolic` у которого пик находится не прямо при количестве воркеров равному количеству ядер, а на 5-6 , при том в дальнейшем при увеличении воркеров показатель также начинает уменьшаться.

Анализ производительности cache

В данном разделе мы рассмотрим параметры `cache-level` и `l1cache-ways` для `cache`.

К сожалению, `perf` не поддерживает большую часть счётчиков на вм:

```
joulin@matthewserver:~$ sudo perf stat -B -e cache-references,cache-misses,cycles,instructions,branches sleep 10s
Performance counter stats for 'sleep 10s':
                                          cache-references
                                          cache-misses
                                          cycles
                                          instructions
                                          branches

10.006422625 seconds time elapsed

0.002162000 seconds user
0.000000000 seconds sys
```

Так что будем использовать метрики из stress-ng и будем отслеживать показатель BOGO OPS.

Команды для анализа с изменением количества воркеров:

```
stress-ng --cache N --cache-level lvl -t 10s --metrics  
stress-ng --cache N --l1cache-ways W -t 10s --metrics
```

Где N – количество воркеров, lvl - уровень кэша (1 - 3), W - количество путей

Напишем скрипты на языке Bash для тестирования:

cache-level

```
#!/bin/bash  
CACHE_LEVEL_START=1  
CACHE_LEVEL_END=3  
CACHE_WORKERS_START=1  
CACHE_WORKERS_END=5  
  
echo "Analize cache-levels"  
  
for lvl in $(seq $CACHE_LEVEL_START $CACHE_LEVEL_END)  
do  
    echo -e "\nProcess cache-level $lvl"  
    for i in $(seq $CACHE_WORKERS_START $CACHE_WORKERS_END)  
    do  
        echo -e "\nStart $i worker(s)\n"  
        stress-ng --cache "$i" --cache-level "$lvl" -t 10s --metrics  
    done  
done
```

l1cache-ways

```
#!/bin/bash
CACHE_WORKERS_START=1
CACHE_WORKERS_END=5

echo "Analize l1cache-ways"

for w in 1 2 4 8 16
do
    echo -e "\nProcess l1cache-ways $w"
    for i in $(seq $CACHE_WORKERS_START $CACHE_WORKERS_END)
    do
        echo -e "\nStart $i worker(s)\n"
        stress-ng --cache "$i" --l1cache-ways "$w" -t 10s --metrics
    done
done
```

Результаты работы можно увидеть по ссылкам:

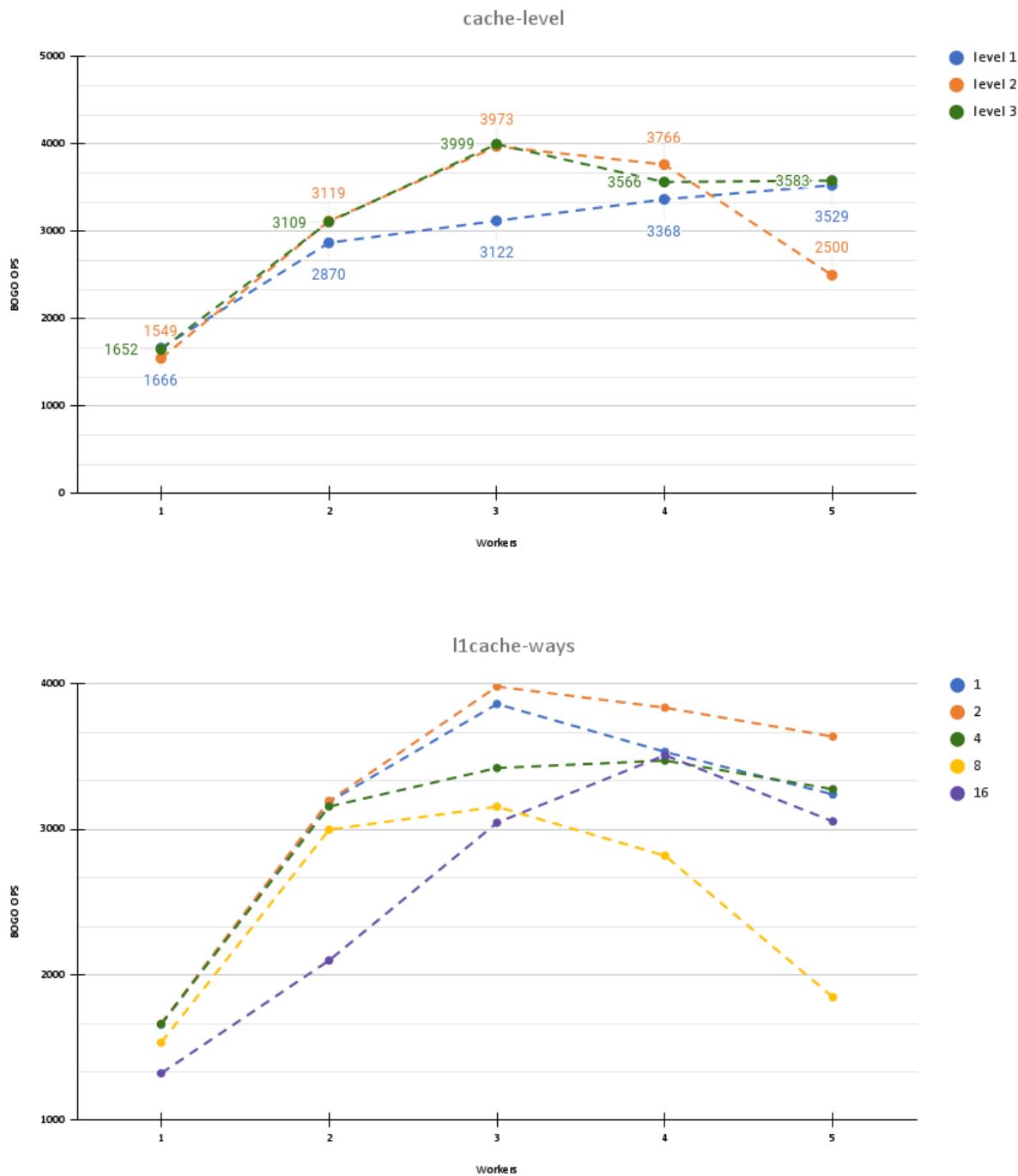
Для cache-level: <https://pastebin.com/85mqTdyB>

Для l1cache-ways: <https://pastebin.com/bTxT9SWb>

Проанализируем результаты:

Workers	cache-level			l1cache-ways				
	1	2	3	1	2	4	8	16
1	1666	1549	1652	1658	1665	1663	1535	1324
2	2870	3119	3109	3190	3197	3158	2999	2100
3	3122	3973	3999	3863	3983	3423	3157	3047
4	3368	3766	3566	3534	3838	3474	2820	3511
5	3529	2500	3583	3241	3639	3277	1847	3056

Графики:



Как видно, использование l1/l2/l3 кэша не так сильно влияет на производительность системы, но l1 показал себя более стабильным. В случае же l1cache-ways мы наблюдаем немнога разнящиеся значения, по большей части вызванные ненадежностью самой метрики bogo ops. Но всё равно, мы видим, что лучшие показатели производительности мы получили, при равных l1cache-ways 2-4, что совпадает с реальными значениями l1cache-ways для моей вм.

Анализ производительности IO

Анализировать будем iomix и ioprio, а мониторить показатели будем изменяя количество воркеров. Для сбора данных, возьмем еще показатели sar 1 10 -u -b.

Команды для анализа с изменением количества воркеров:

```
stress-ng --iomix N -t 15s --metrics | sar 1 10 -u -b  
stress-ng --ioprio N -t 15s --metrics | sar 1 10 -u -b
```

Где N – количество воркеров

Напишем скрипт на языке Bash для тестирования:

```
#!/bin/bash  
WORKERS_START=1  
WORKERS_END=8  
TARGET="$1"  
  
if [ -z $TARGET ]  
then  
    echo "Pass target IO method"  
else  
    echo "Analize $TARGET"  
    for i in $(seq $WORKERS_START $WORKERS_END)  
    do  
        echo -e "\nstart $i worker(s)\n";  
        stress-ng --$TARGET $i -t 15s --metrics | sar 1 10 -u -b  
    done  
fi
```

Результаты работы можно увидеть по ссылкам:

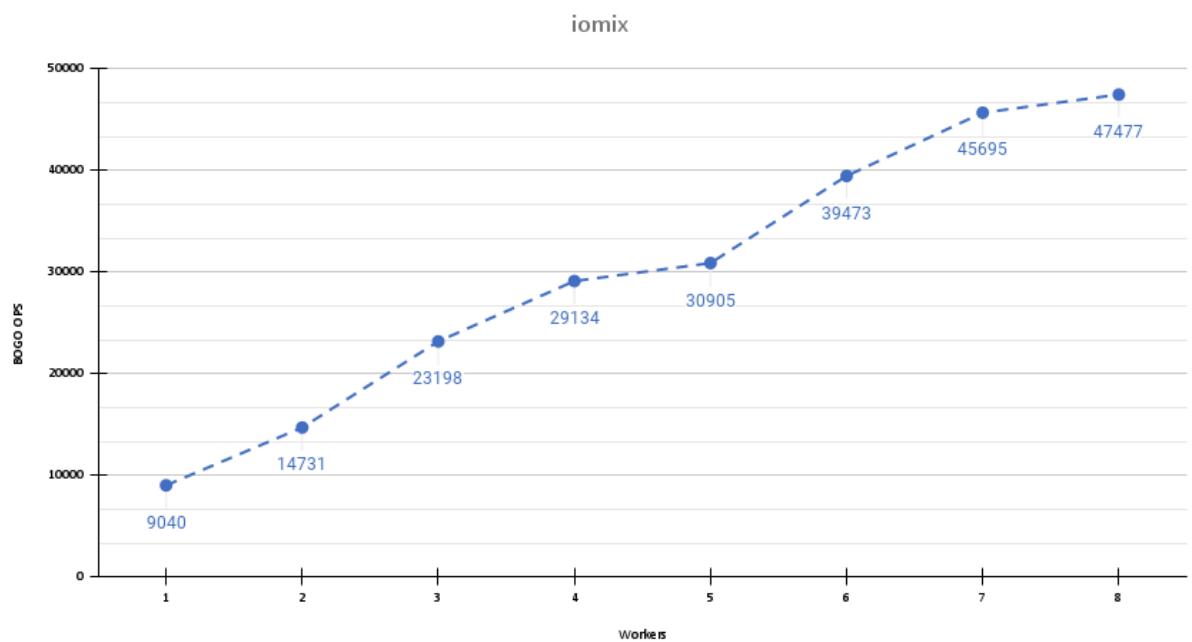
Для iomix: <https://pastebin.com/dUgF97mn>

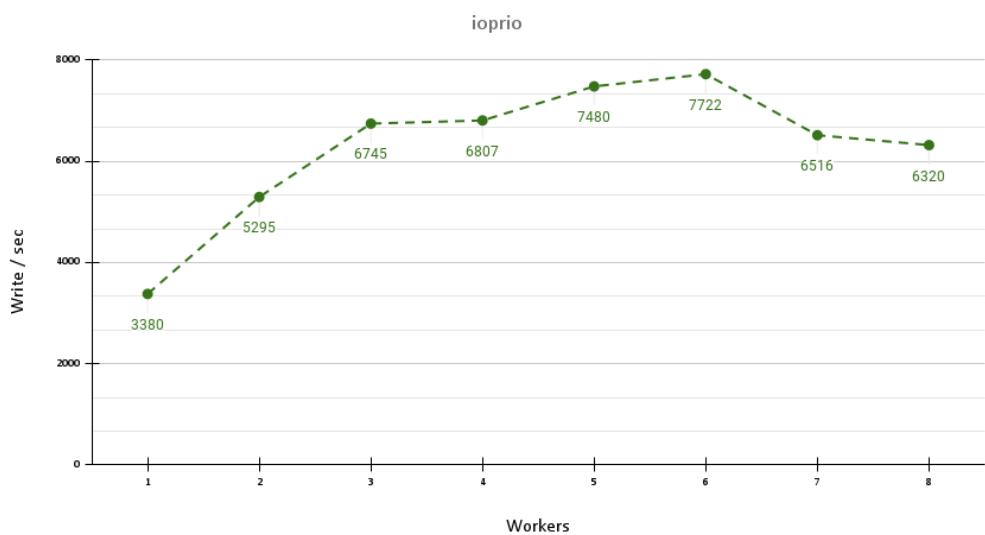
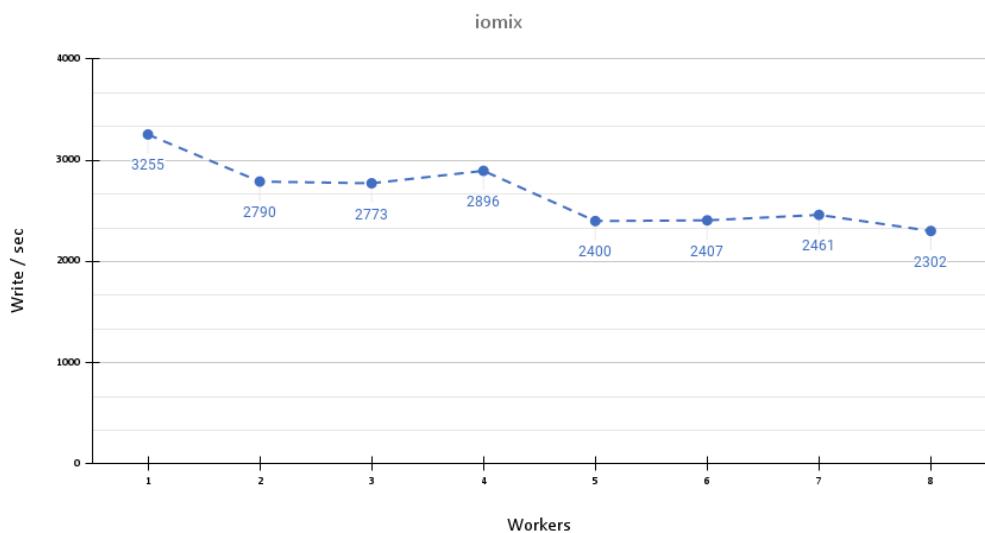
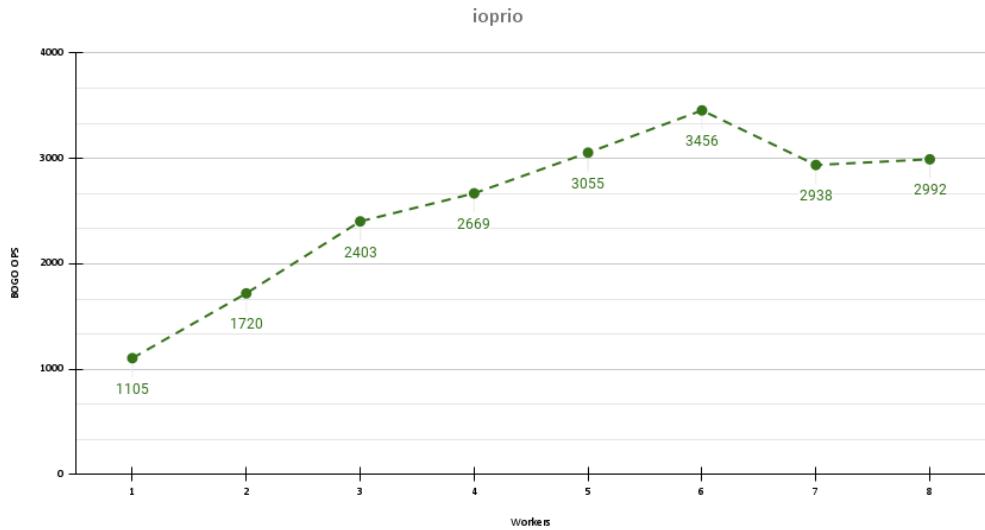
Для ioprio: <https://pastebin.com/5lyKshj8>

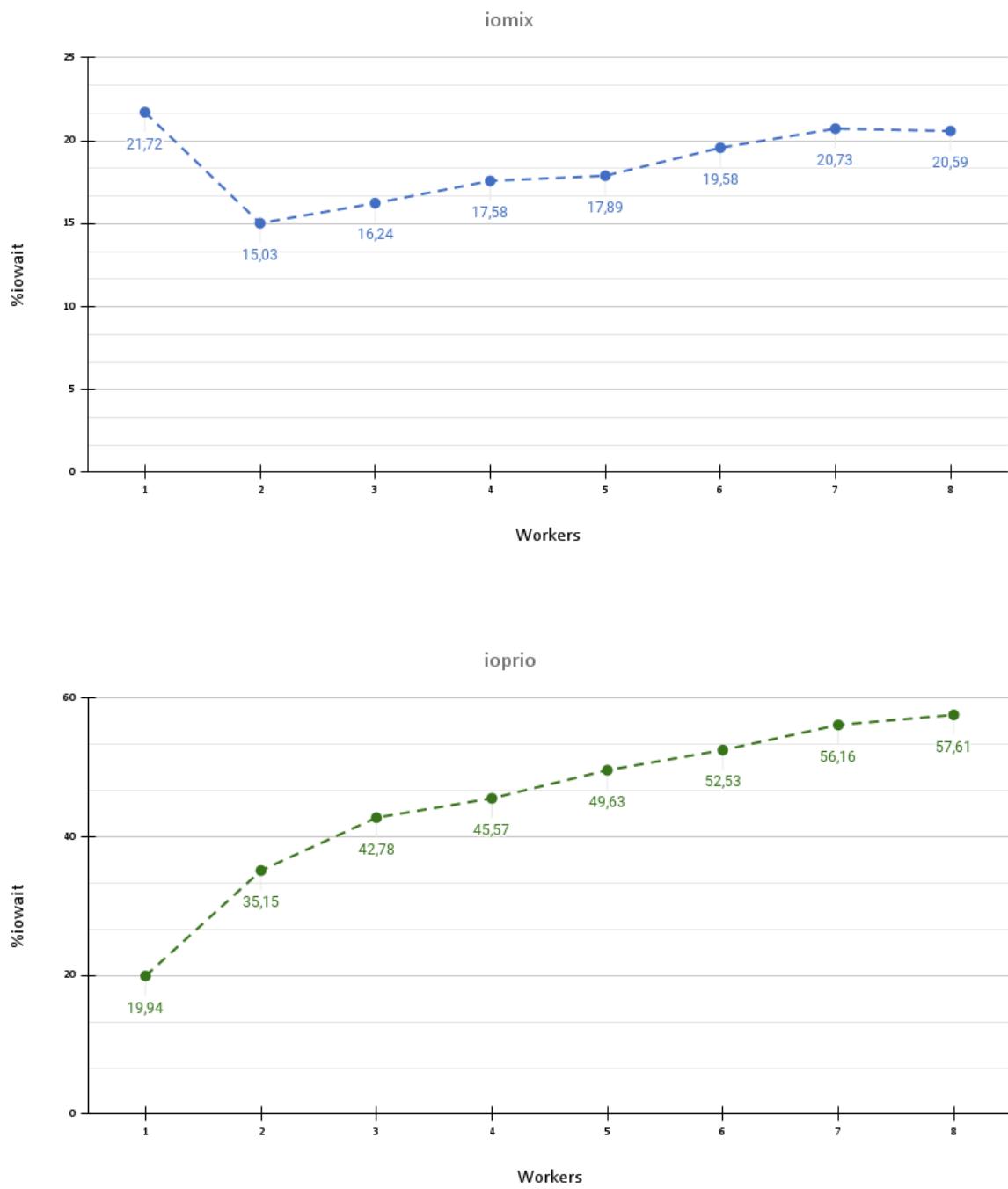
Проанализируем результаты:

Workers	iomix			ioprio		
	bogops	w/s	avg %iowait	bogops	w/s	avg %iowait
1	9040	3255	21,72	1105	3380	19,94
2	14731	2790	15,03	1720	5295	35,15
3	23198	2773	16,24	2403	6745	42,78
4	29134	2896	17,58	2669	6807	45,57
5	30905	2400	17,89	3055	7480	49,63
6	39473	2407	19,58	3456	7722	52,53
7	45695	2461	20,73	2938	6516	56,16
8	47477	2302	20,59	2992	6320	57,61

Графики:







Как видно, мы наблюдаем повышение B0G0 OPS при увеличении количества воркеров, но максимум достигается при количестве воркеров равному удвоенному значению количества ядер процессора. TPS/wtps же для **iomix** немного уменьшался с увеличением количества воркеров, при том для **ioprio** он вел себя ровно наоборот. **%iowait** в обоих случаях рос, так как очевидно повышалось количество IO операций при увеличении воркеров, но в случае **ioprio** он показал более быстрый рост.

Анализ производительности memory

Анализировать будем mcontend и shm, а мониторить показатели будем изменяя количество воркеров. Для сбора данных, возьмем еще показатели sar 1 10 -r.

Команды для анализа с изменением количества воркеров:

```
stress-ng --mcontend N -t 15s --metrics | sar 1 10 -r  
stress-ng --shm N -t 15s --metrics | sar 1 10 -r
```

Где N – количество воркеров

Напишем скрипт на языке Bash для тестирования:

```
#!/bin/bash  
WORKERS_START=1  
WORKERS_END=12  
TARGET="$1"  
  
if [ -z $TARGET ]  
then  
    echo "Pass target Memory method"  
else  
    echo "Analize $TARGET"  
    for i in $(seq $WORKERS_START $WORKERS_END)  
    do  
        echo -e "\nstart $i worker(s)\n";  
        stress-ng --$TARGET $i -t 15s --metrics | sar 1 10 -r  
    done  
fi
```

Результаты работы можно увидеть по ссылкам:

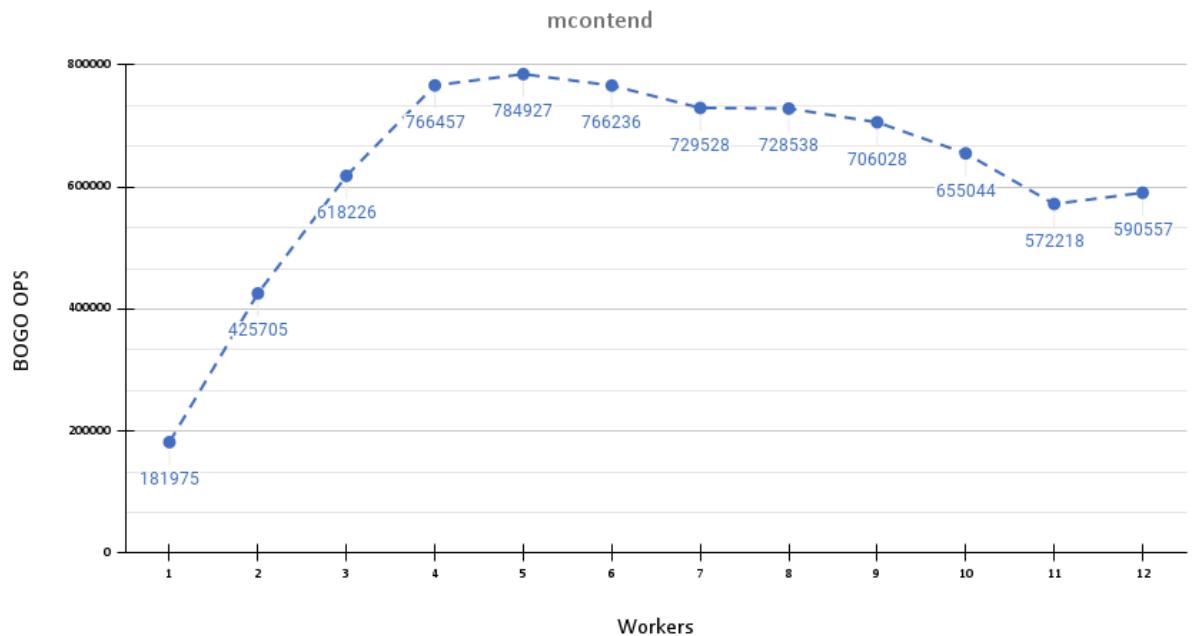
Для mcontend: <https://pastebin.com/9yCiAPtF>

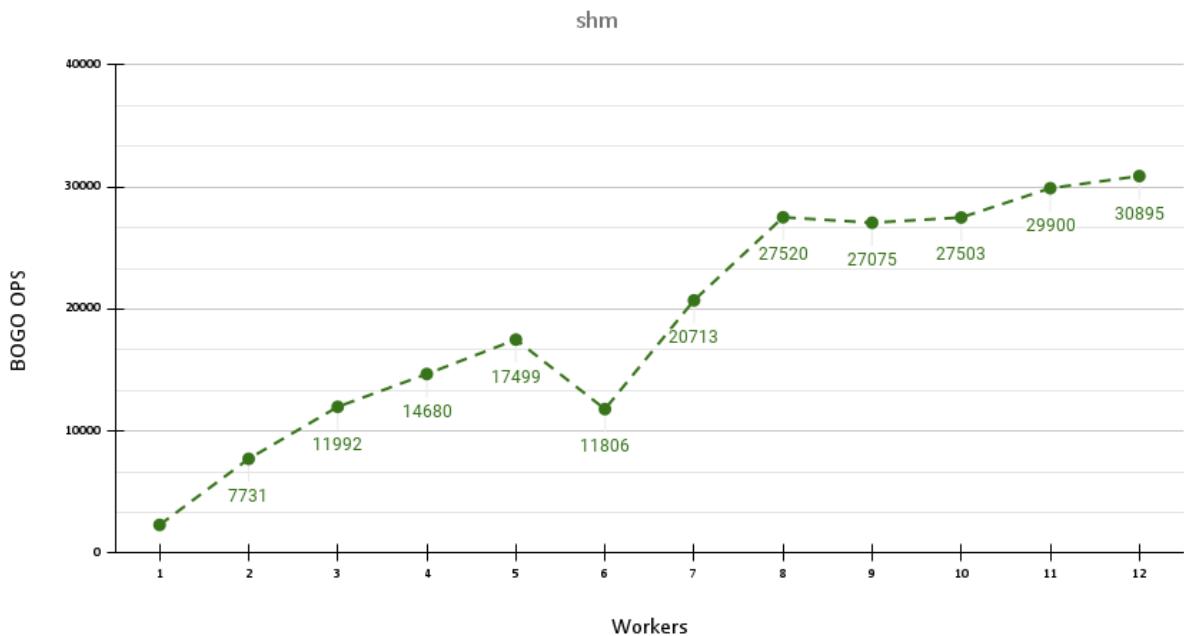
Для shm: <https://pastebin.com/2523fXNW>

Проанализируем результаты:

Workers	mcontend		shm	
	bogops	%commit	bogops	%commit
1	181975	8,68	2320	16
2	425705	9,42	7731	15
3	618226	10,16	11992	16,25
4	766457	10,9	14680	16,35
5	784927	11,64	17499	16,12
6	766236	12,38	11806	17,07
7	729528	13,13	20713	17,39
8	728538	13,87	27520	18,13
9	706028	14,61	27075	19,01
10	655044	15,35	27503	21,87
11	572218	16,09	29900	22,43
12	590557	16,83	30895	23,15

Графики:





Как видно, показатели ведут себя по-разному при увеличении количества воркеров. `mcontend` достигает своего пика при количестве воркеров примерно равному количеству ядер, в то время как `shm` показывает постоянный постепенный рост (но и бывают единичные моменты проседания метрики).

Анализ производительности `network`

Анализировать будем `netlink-proc` и `netlink-task`, а мониторить показатели будем изменяя количество воркеров. Других показателей, кроме BOGO OPS, я не нашёл, так как `netlink-proc` создаёт дочерние процессы, события которых мониторятся и пишутся в BOGO OPS, а `netlink-task` вызывает сбор статистики у другой утилиты.

Команды для анализа с изменением количества воркеров:

```
stress-ng --netlink-proc N -t 10s --metrics
stress-ng --netlink-task N -t 10s --metrics
```

Где N – количество воркеров

Напишем скрипт на языке Bash для тестирования:

```
#!/bin/bash
WORKERS_START=1
WORKERS_END=10
TARGET="$1"

if [ -z $TARGET ]
then
    echo "Pass target Network method"
else
    echo "Analize $TARGET"
    for i in $(seq $WORKERS_START $WORKERS_END)
    do
        echo -e "\nstart $i worker(s)\n";
        stress-ng --$TARGET $i -t 10s --metrics
    done
fi
```

Результаты работы можно увидеть по ссылкам:

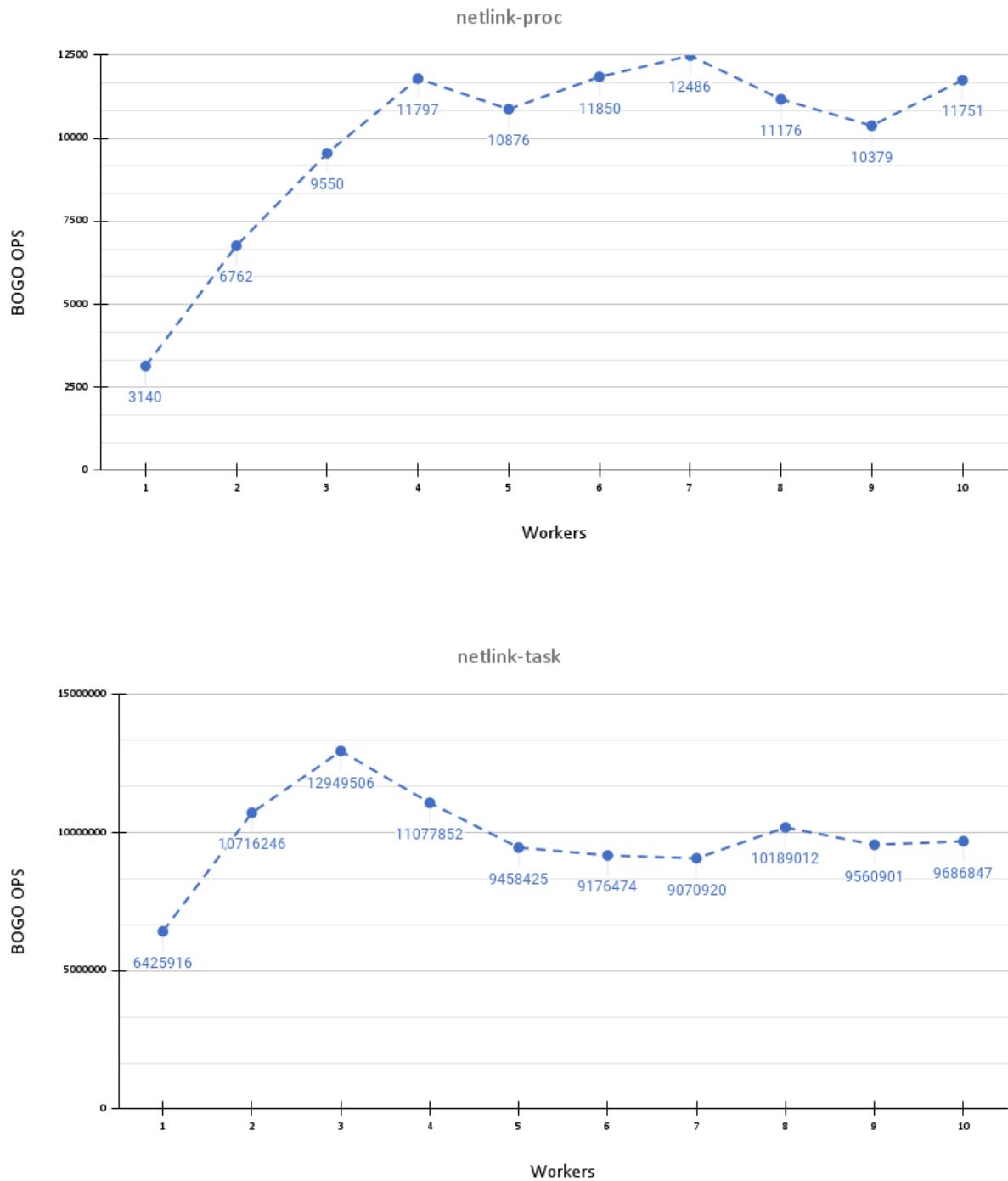
Для netlink-proc: <https://pastebin.com/9AFZH6Cy>

Для netlink-task: <https://pastebin.com/3HYzEwQ3>

Проанализируем результаты:

Workers	netlink-proc		netlink-task	
	bogops	bogo ops / s	bogops	bogo ops / s
1	3140	314	6425916	642562
2	6762	676	10716246	1071610
3	9550	955	12949506	1294918
4	11797	1180	11077852	1107773
5	10876	1088	9458425	945805
6	11850	1185	9176474	917574
7	12486	1249	9070920	907069
8	11176	1118	10189012	1018640
9	10379	1038	9560901	956026
10	11751	1175	9686847	968171

Графики:



Как видно, оба параметра растут до момента, пока количество воркеров не станет равным количеству ядер процессора (а в случае netlink-task даже чуть раньше), а затем немного падают (как в netlink-task) и остаются приблизительно на этом значении.

Анализ производительности pipe

Анализировать будем `pipeherd-yield` и `sigpipe`, а мониторить показатели будем изменяя количество воркеров. Так как `sigpipe` является воркером, а `pipeherd-yield` просто флагом, который принудительно выполняет планирование после каждой записи, то проверим, как он повлияет на исполнение нашего воркера (B0GO OPS и context-switches). Также будем смотреть на нагрузку на CPU с помощью `sar 1 10 -u`.

Команды для анализа с изменением количества воркеров:

```
perf stat -e cs stress-ng --sigpipe N -t 10s --metrics | sar 1 10 -u  
perf stat -e cs stress-ng --sigpipe N --pipeherd-yield -t 10s  
--metrics | sar 1 10 -U
```

Где N – количество воркеров

Напишем скрипт на языке Bash для тестирования:

```
#!/bin/bash  
WORKERS_START=1  
WORKERS_END=10  
TARGET=sigpipe  
  
echo "Analize $TARGET"  
for i in $(seq $WORKERS_START $WORKERS_END)  
do  
    echo -e "\nstart $i worker(s) without flag\n"  
    perf stat -e cs stress-ng --$TARGET $i -t 10s --metrics | sar 1 10  
-u  
    echo -e "\nstart $i worker(s) with flag\n"  
    perf stat -e cs stress-ng --$TARGET $i --pipeherd-yield --metrics | sar 1  
10 -u  
done
```

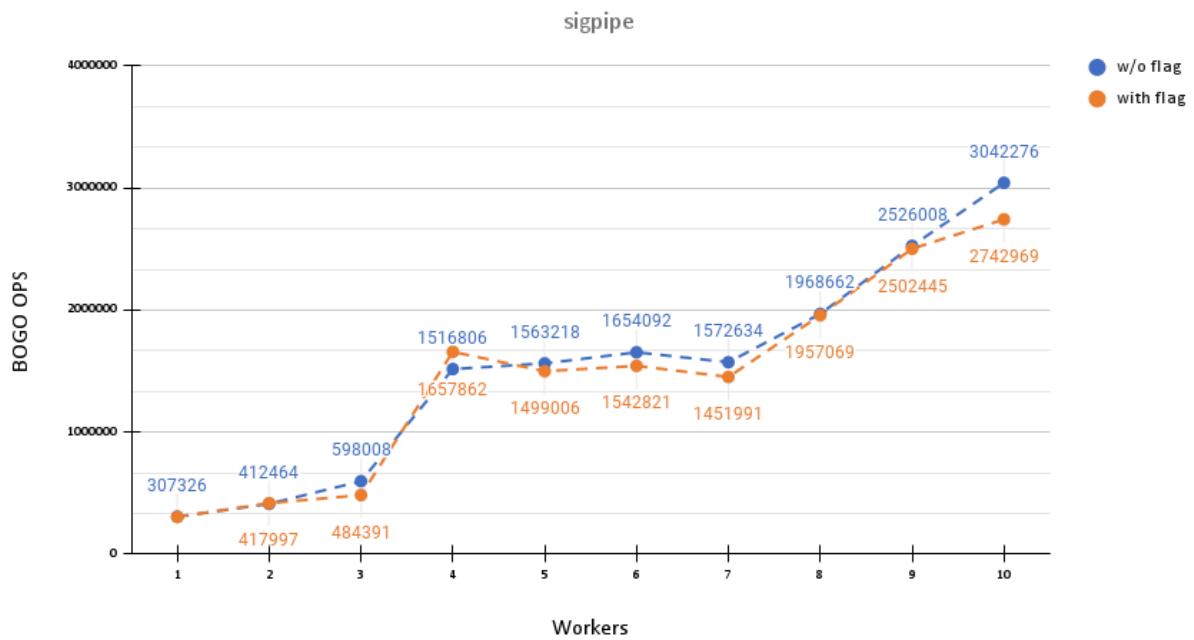
Результаты работы можно увидеть по ссылке:

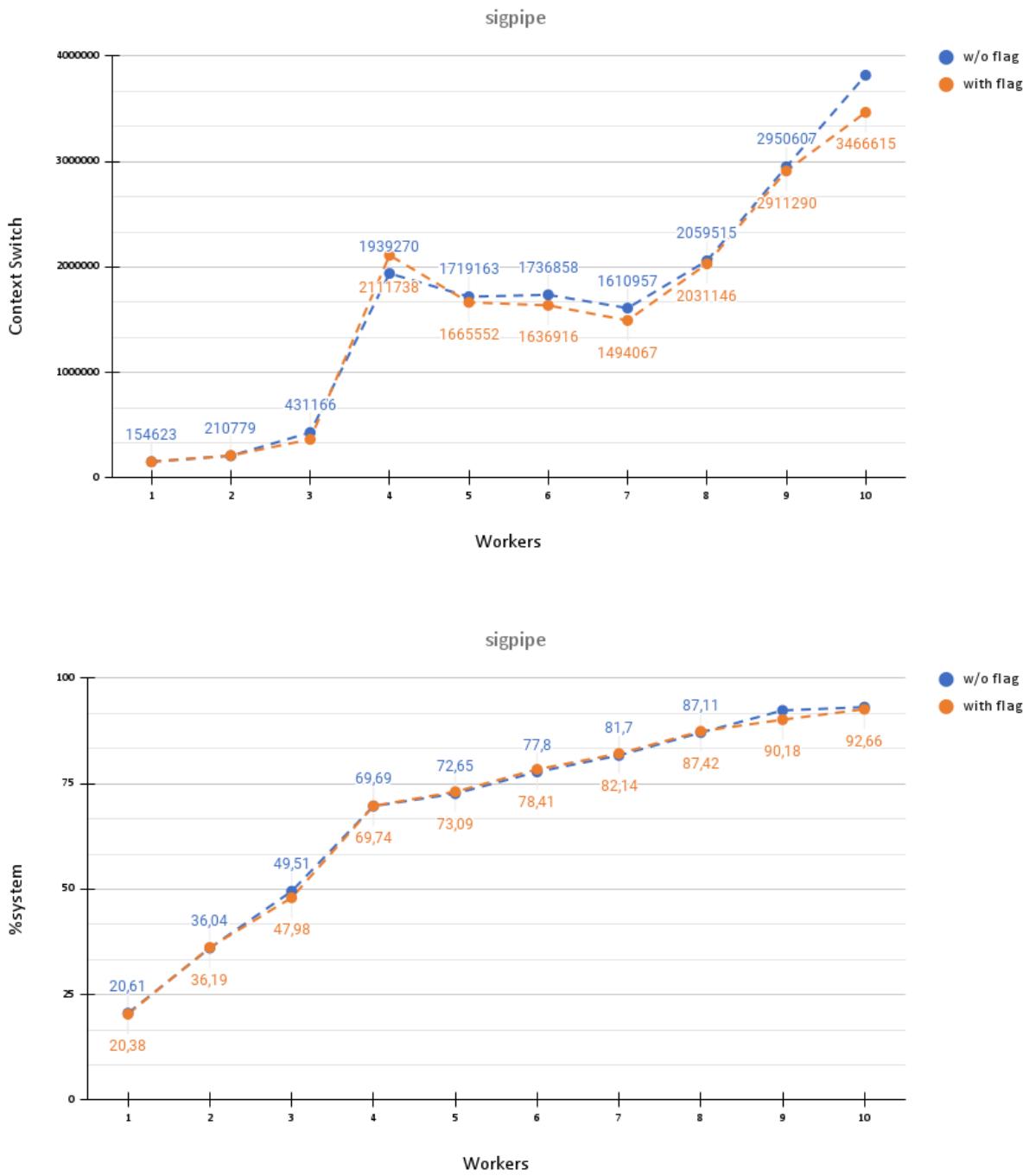
<https://pastebin.com/V2iktWJL>

Проанализируем результаты:

Workers	sigpipe w/o flag			sigpipe with flag		
	bogops	cs	avg %system	bogops	cs	avg %system
1	307326	154623	20,61	305496	153641	20,38
2	412464	210779	36,04	417997	213109	36,19
3	598008	431166	49,51	484391	366303	47,98
4	1516806	1939270	69,69	1657862	2111738	69,74
5	1563218	1719163	72,65	1499006	1665552	73,09
6	1654092	1736858	77,8	1542821	1636916	78,41
7	1572634	1610957	81,7	1451991	1494067	82,14
8	1968662	2059515	87,11	1957069	2031146	87,42
9	2526008	2950607	92,37	2502445	2911290	90,18
10	3042276	3818690	93,16	2742969	3466615	92,66

Графики:





Как видно по результатам, флаг pipeherd-yield не влияет на производительность системы при работе `sigpipe`. Но мы можем наблюдать, что при количестве вореков, подходящему к утроенному значению ядер процессора, мы получаем почти 100% загрузку `%system` и пик производительности. В данном примере, возможно, флаг не был полезен, так как `sigpipe` выполняет системные вызовы `clone` и `fork`.

Анализ производительности sched

Анализировать будем sched-period и schedpolicy, а мониторить показатели будем изменяя количество воркеров. Для мониторинга schedpolicy будем также снимать показатели нагрузки системы с помощью sar 1 10 -u -b. А так как параметр sched-period просто конфигурирует deadline scheduler, то для анализа рассмотрим yield, который запускает воркеры.

Команды для анализа с изменением количества воркеров:

```
stress-ng --schedpolicy N -t 10s --metrics | sar 1 10 -u -b  
  
perf stat -e cs stress-ng --yield N --sched deadline --sched-period  
per -t 10s --metrics | sar 1 10 -u -b
```

Где N – количество воркеров, per - параметр периода для deadline (в нс)

Напишем скрипты на языке Bash для тестирования:

schedpolicy.sh

```
#!/bin/bash  
WORKERS_START=1  
WORKERS_END=10  
TARGET=schedpolicy  
  
echo "Analize $TARGET"  
for i in $(seq $WORKERS_START $WORKERS_END)  
do  
    echo -e "\nstart $i worker(s)\n"  
    stress-ng --$TARGET $i -t 10s --metrics | sar 1 10 -u -b  
done
```

sched_deadline.sh

```
#!/bin/bash
WORKERS_START=1
WORKERS_END=4
TARGET=sched-period

echo "Analize $TARGET"
for i in $(seq $WORKERS_START $WORKERS_END)
do
    echo -e "\nstart $i worker(s)\n"
    for per in 100 10000 1000000
    do
        echo -e "\nUse period $per ns\n"
        perf stat -e cs stress-ng --yield $i --sched deadline --$TARGET $per
-t 10s --metrics | sar 1 10 -u -b
    done
done
```

Результаты работы можно увидеть по ссылке:

Для sched-period: <https://pastebin.com/MX5xFPcj>

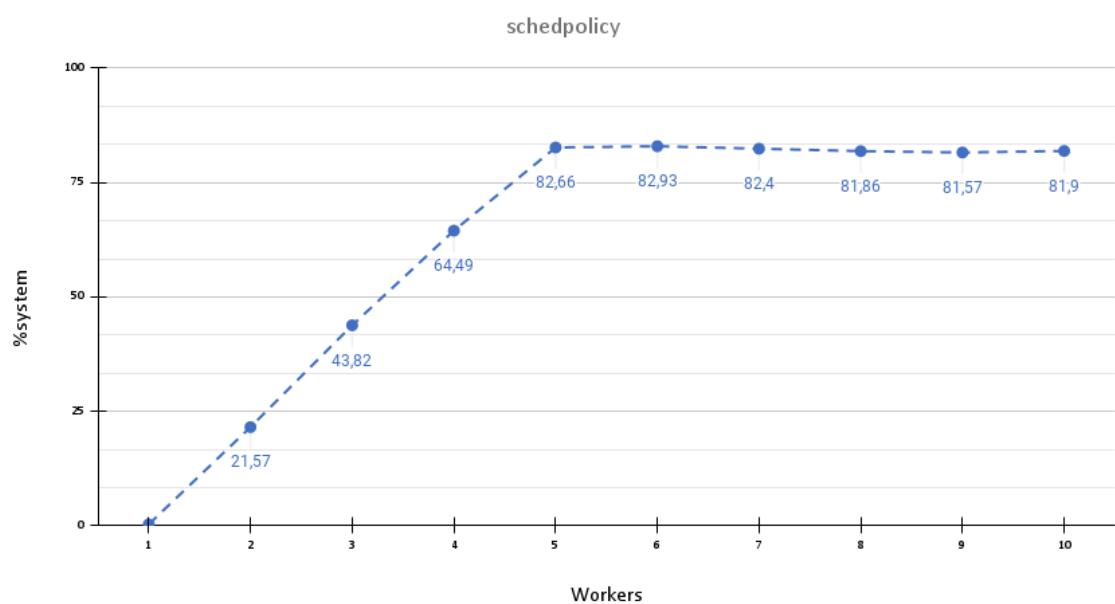
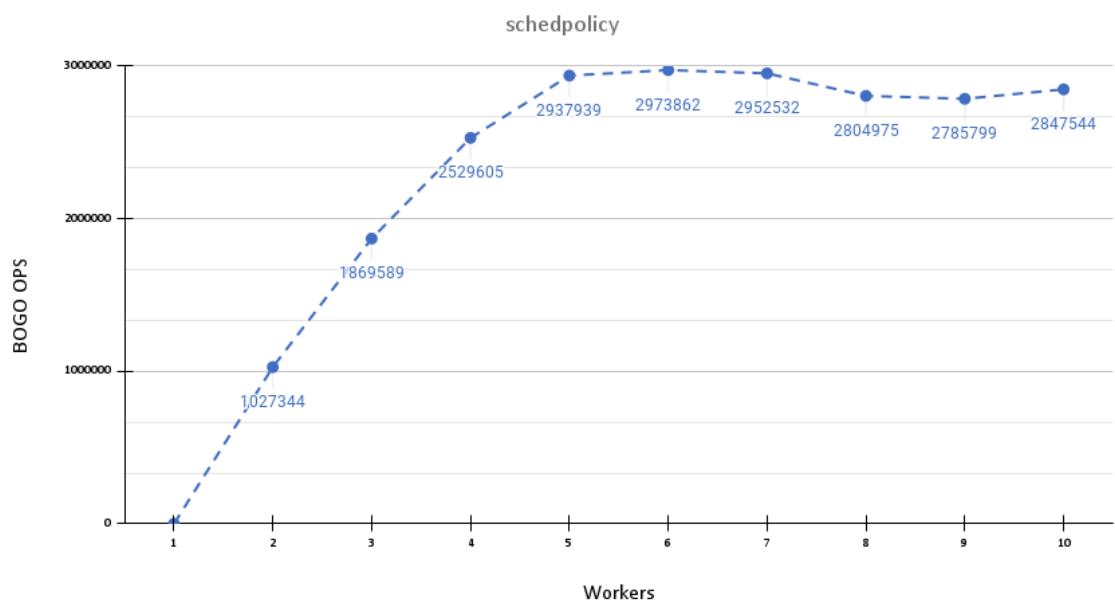
Для schedpolicy: <https://pastebin.com/CFVk4wS6>

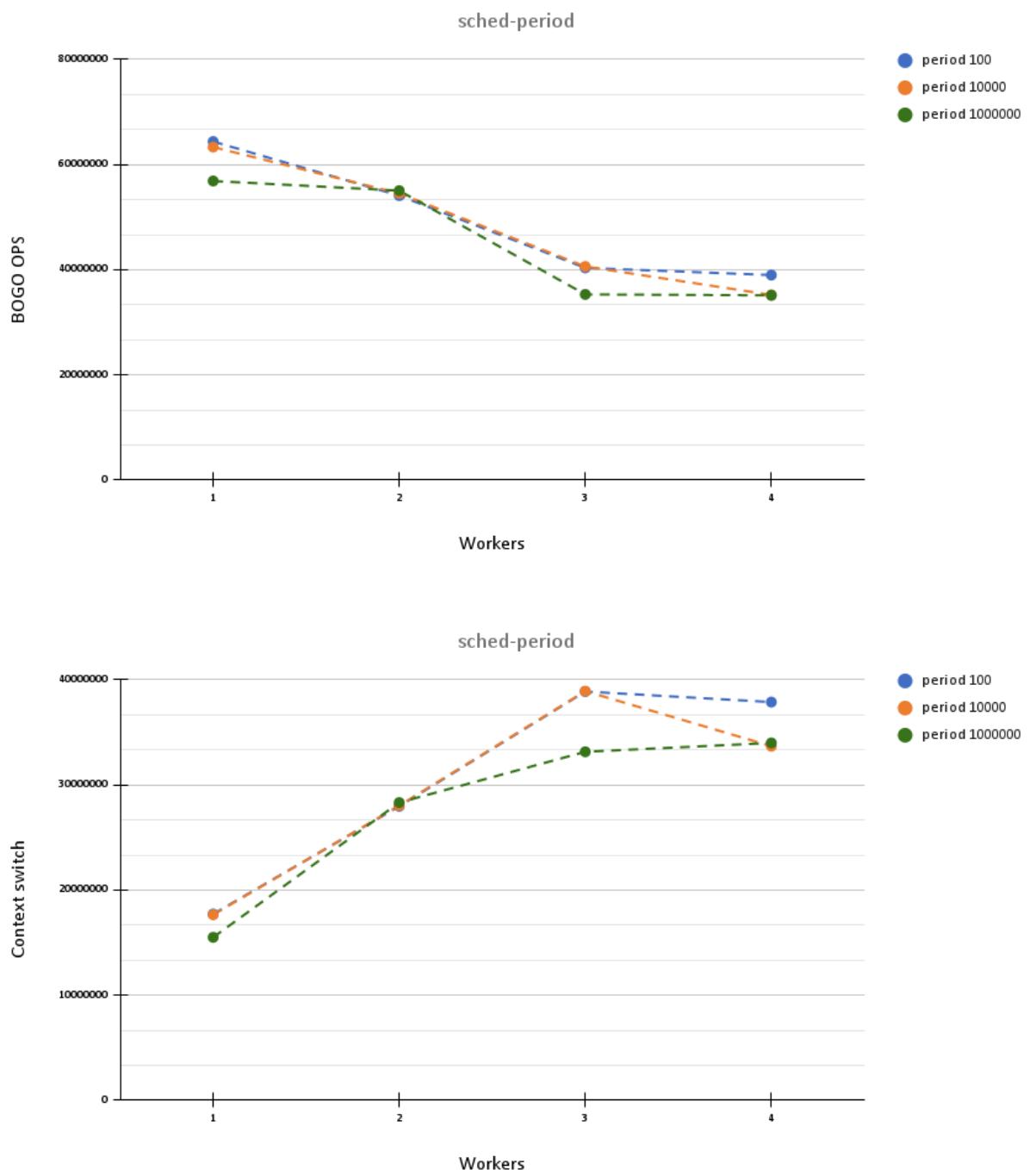
Проанализируем результаты:

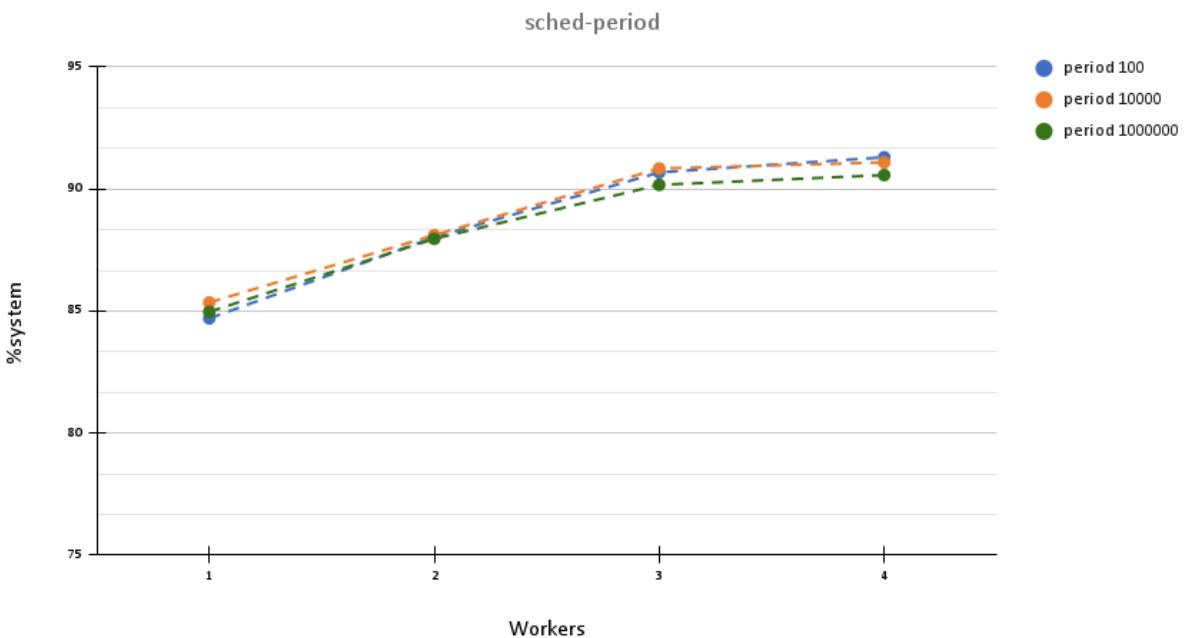
schedpolicy		
Workers	bogops	avg %system
1	241	0,33
2	1027344	21,57
3	1869589	43,82
4	2529605	64,49
5	2937939	82,66
6	2973862	82,93
7	2952532	82,4
8	2804975	81,86
9	2785799	81,57
10	2847544	81,9

sched-period									
period	100			10000			1000000		
Workers	bogops	cs	avg %system	bogops	cs	avg %system	bogops	cs	avg %system
1	64378351	17718557	84,7	63329733	17655521	85,35	56875835	15506385	84,97
2	54093969	27966376	88,02	54511040	28029315	88,11	55039966	28319424	87,96
3	40350288	38877222	90,68	40620416	38915347	90,84	35307548	33146252	90,17
4	38986339	37873444	91,3	35207481	33679285	91,09	35132052	33988214	90,57

Графики:







Для `schedpolicy` по результатам видно, что пиковая производительность достигается при количестве воркеров равному количеству ядер + 1. Для параметра же `sched-period` можно заметить, что увеличение этого параметра ведёт к ухудшению производительности (проседает BOGO OPS, cs, %system), но и в целом при увеличении числа воркеров `yield` производительность только падает, растёт только количество context-switch-ей.

Заключение

Интересная лабораторная работа, которая позволила покопаться в различных метриках для измерения производительности разных подсистем моего компьютера. Да, занятие это довольно нудное, так как нужно собирать всё кропотливо руками и в жизни я бы таким не занимался, но в контексте ЛР мне пришлось и местами было интересненько.