

# Penjadwalan *Garbage Collection* pada RAID SSD untuk Meminimasi *Collision*

**Dosen Pembimbing:** Achmad Imam Kistijantoro ST, M.Sc., Ph.D. dan Riza Satria Perdana ST, MT.

---



**Fadhil Imam Kurnia**

**13515146**

IF4092 Tugas Akhir II

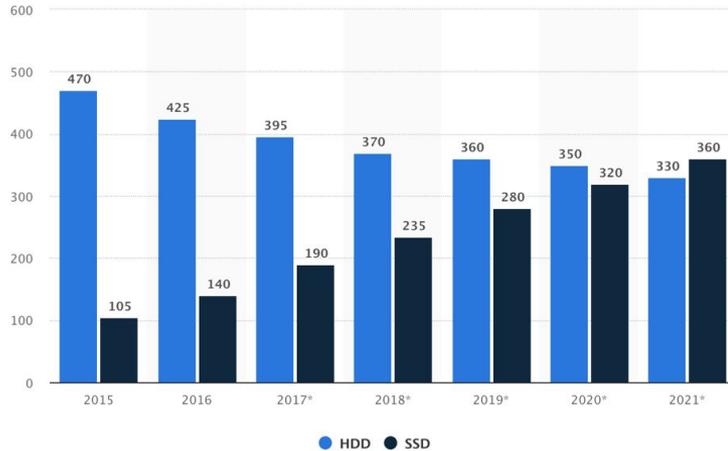
# Alur Presentasi

- ❑ Latar Belakang
- ❑ Tujuan
- ❑ Batasan
- ❑ Metodologi
- ❑ Analisis & Rancangan
- ❑ Pengujian
- ❑ Kesimpulan

1.

# Latar Belakang

**Tren Jumlah Pengiriman HDD dan SSD dari Tahun 2015-2021**  
(dalam juta)



Sumber: Statista.com

## Jumlah pengiriman SSD meningkat!

Jumlah pengiriman SSD pada tahun 2016 adalah **140 juta item**, diprediksikan jumlah tersebut akan terus bertambah. Sedangkan jumlah pengiriman HDD diprediksikan semakin turun.

**Pada tahun 2021 diprediksikan jumlah pengiriman SSD sudah melampaui HDD.**  
(Statista.com, 2017)

Beberapa keunggulan SSD dibanding HDD:

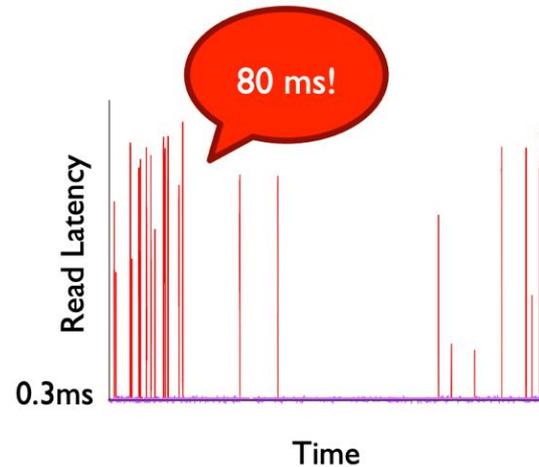
- Tidak ada komponen bergerak
- Kecepatan akses yang lebih tinggi
- Kepadatan ruang penyimpanan

# Namun ada masalah tersembunyi pada SSD

## Proses GC dapat mengakibatkan perlambatan!

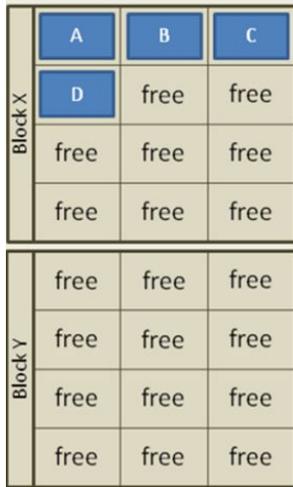
Pada SSD yang hampir penuh proses *Garbage Collection* (GC) akan sering muncul, hal tersebut dapat mengakibatkan proses pembacaan data menjadi lambat.

**Latensi proses pembacaan dapat meningkat hingga 200x lipat!** (Yan, 2017)

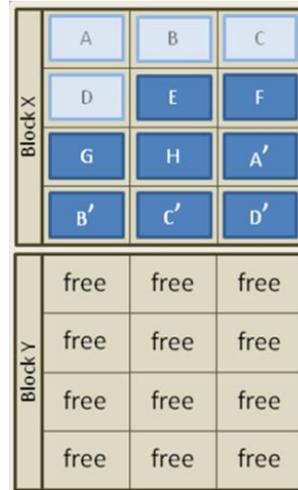


# Apa itu proses GC dalam SSD?

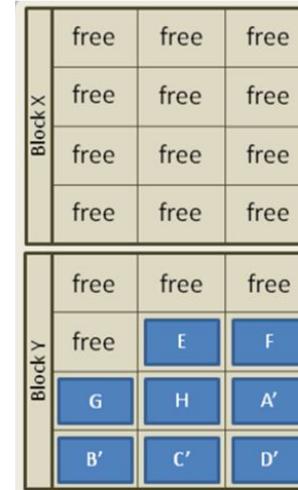
GC secara umum adalah proses pembebasan memori yang ditempati oleh objek yang sudah tidak digunakan lagi.



Penulisan data (A-D) pada *page* dalam *block X*



Penambahan data (E-H) dan modifikasi (A-D) menjadi (A'-D')  
Data lama (A-D) tidak dihapus, tapi ditandai sebagai *invalid page*.



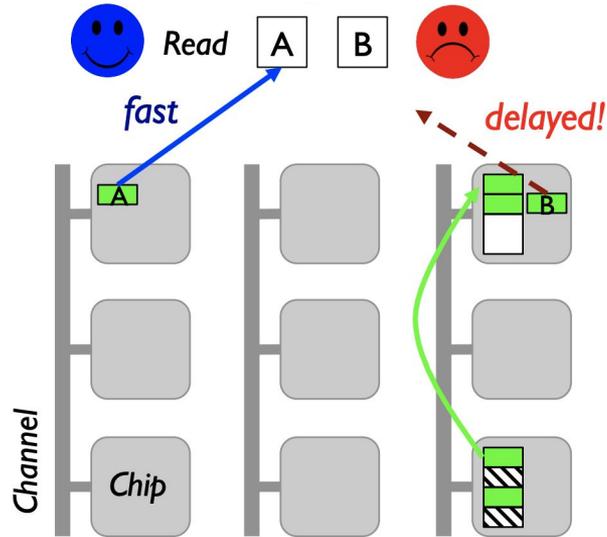
## Proses Garbage Collection

Read dan write dilakukan pada level *page*, penghapusan pada level *block*. Perlu memindahkan *valid page* sebelum menghapus sebuah *block*.

Keterangan:



# Bagaimana GC menghambat proses *I/O read* ?

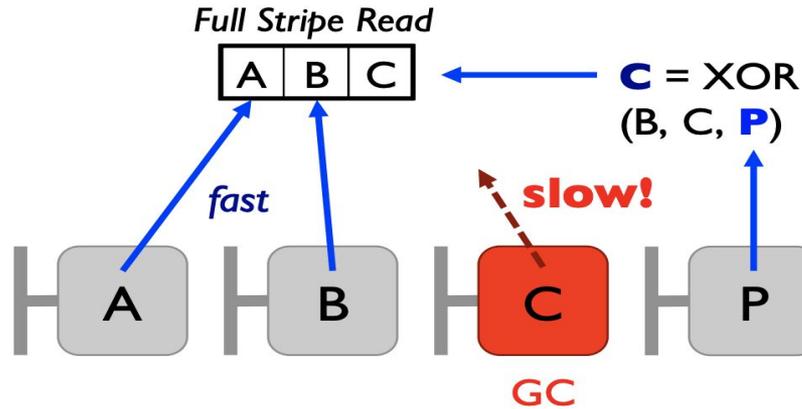


Proses GC memindahkan banyak *valid page* sekaligus.

Ini membuat SSD sibuk menangani proses GC selama puluhan ms!

Arsitektur Internal SSD dan proses GC di dalamnya  
(Yan, 2017)

# Penelitian sebelumnya



Penelitian ttFlash (Yan, 2017) memanfaatkan paritas untuk merekonstruksi data yang lambat dibaca karena proses GC, perlu mengimplementasikan proses *rotating GC* untuk meminimasi *collision* antar GC.

**Hal tersebut dapat mengeliminasi dampak dari GC**

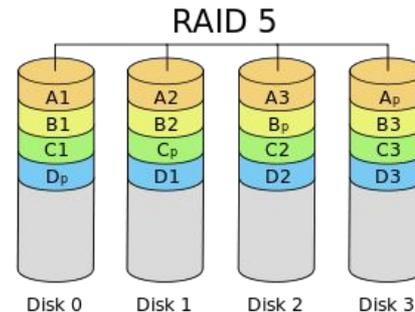
# Kita dapat menerapkannya juga pada **RAID SSD!**

Efek dari proses GC lebih parah pada RAID SSD karena proses *read* dilakukan antar SSD.

Arsitektur internal SSD mirip dengan RAID

Paritas sudah umum digunakan dalam RAID

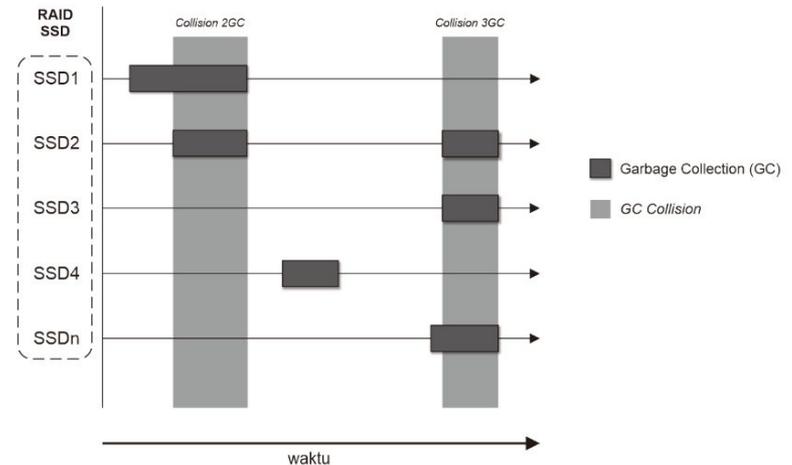
**Perlu mekanisme penjadwalan GC agar *collision* dapat dikurangi!**



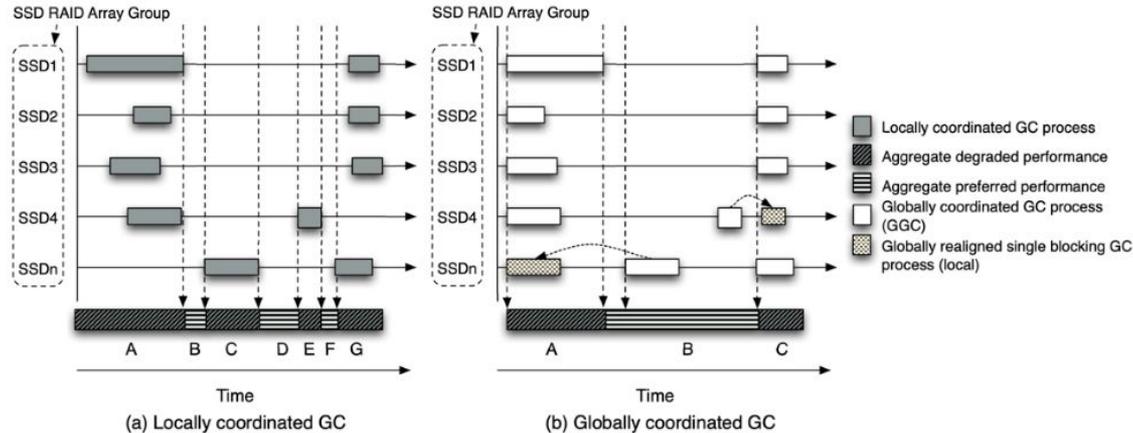
# GC Collision pada RAID SSD

GC Collision atau collision adalah kondisi ketika ada lebih dari 2 proses GC yang terjadi bersamaan pada SSD yang berbeda dalam RAID SSD.

Kondisi ini dapat menghambat pembacaan data pada RAID SSD. Diperlukan penjadwalan GC agar collision dapat dikurangi.



# Penelitian terkait penjadwalan GC pada RAID SSD



Penelitian sebelumnya, **Harmonia**, yang dilakukan oleh Kim sudah berhasil melakukan penjadwalan GC pada RAID SSD. Namun Harmonia justru memaksimumkan *collision* untuk meningkatkan kinerja RAID SSD.

**Sejauh studi literatur yang dilakukan, belum ada penelitian yang khusus meminimasi GC collision pada RAID SSD.** Hal tersebut karena metode rekonstruksi untuk mengeliminasi efek dari GC baru dipublikasikan akhir-akhir ini.

# 2.

# Tujuan

1

**Mendefinisikan strategi yang dapat digunakan untuk dapat meminimasi *GC collision* pada RAID SSD**  
sehingga proses rekonstruksi dapat dilakukan secara optimal ketika salah satu SSD sedang menjalankan proses GC.

2

**Mengimplementasikan strategi minimasi collision yang sudah didefinisikan,**  
serta mengukur efektivitas strategi tersebut.

4.

# Batasan

# Batasan yang digunakan dalam tugas akhir ini



## Menggunakan Simulator

Implementasi minimasi collision hanya dilakukan pada simulator SSD yang sudah divalidasi sebelumnya, atau sudah dipastikan dapat mendekati SSD sesungguhnya.



## Hanya RAID 5

Jenis RAID yang akan digunakan untuk pengujian adalah RAID 5 karena pada jenis tersebut terdapat paritas yang dapat digunakan untuk rekonstruksi data.

# 5. Metodologi

# Metodologi

## Eksplorasi Simulator RAID

Mencoba simulator SSD dan mengecek data apa saja yang bisa dihasilkan

## Perancangan dan Persiapan

Merancang modifikasi simulator yang diperlukan, mekanisme penjadwalan, serta data apa saja yang ingin dianalisis

## Implementasi

Implementasi perancangan pada simulator untuk mensimulasikan RAID SSD dengan GC yang dijadwalkan.

## Pengujian

Menjalankan simulasi RAID SSD dengan penjadwalan, menghasilkan data sebelum dan sesudah implementasi.

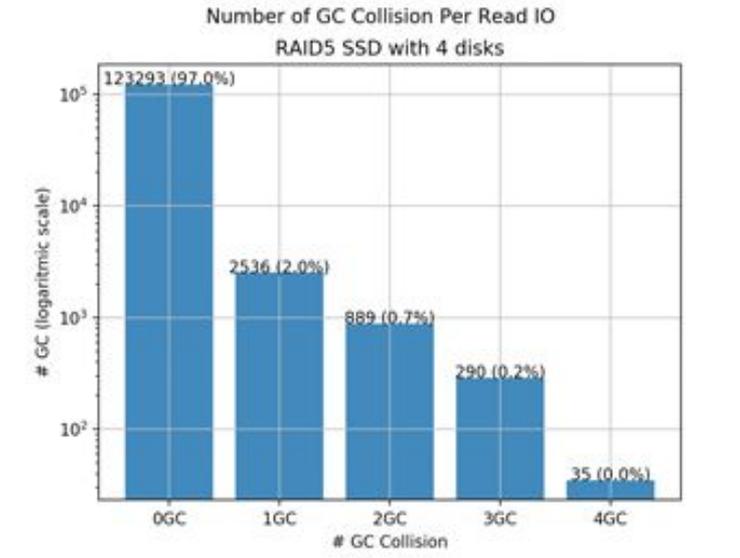
## Penarikan Kesimpulan

Data yang cukup dapat membantu dalam penarikan kesimpulan apakah penjadwalan GC yang dilakukan dapat meminimasi *collision*

6.

# Analisis dan Rancangan

# Kondisi sebelum implementasi



Keterangan sumbu x

0GC : jumlah *read request* yang tidak menjumpai proses GC

1GC: jumlah *read request* yang menjumpai GC pada salah satu disk

2GC: jumlah *read request* yang menjumpai 2 GC pada disk yang berbeda

3GC: jumlah *read request* yang menjumpai 3 GC pada disk yang berbeda

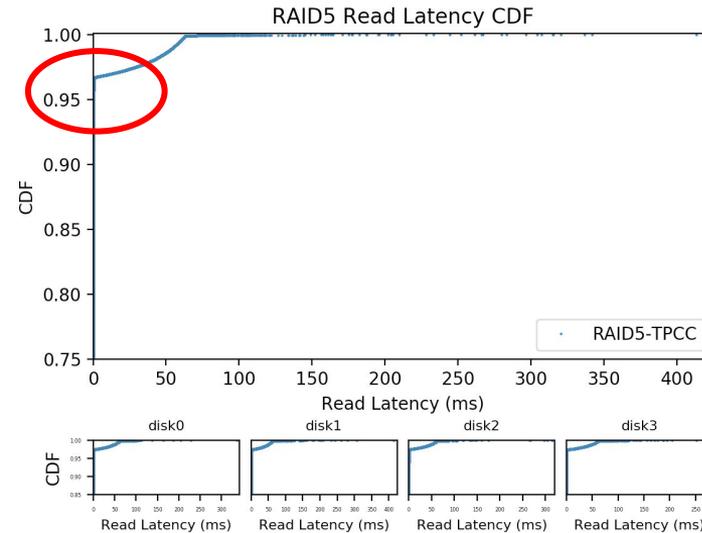
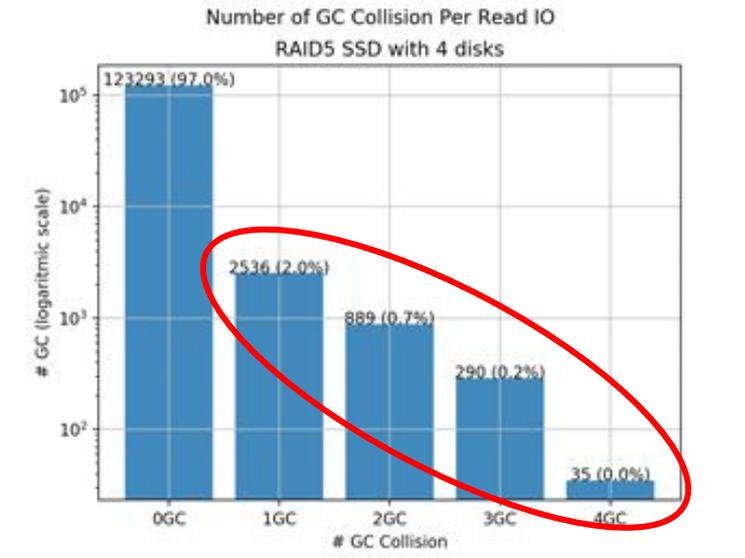
4GC: jumlah *read request* yang menjumpai 4 GC pada disk yang berbeda

...

## Masih banyak GC *collision* yang terjadi!

Jika dilihat pada grafik CDF, jumlah GC dan GC collision sangat berkorelasi terhadap latensi di RAID SSD. Oleh karena itu penting untuk mengeliminasi dampak dari GC, langkah awalnya adalah dengan meminimasi *collision*.

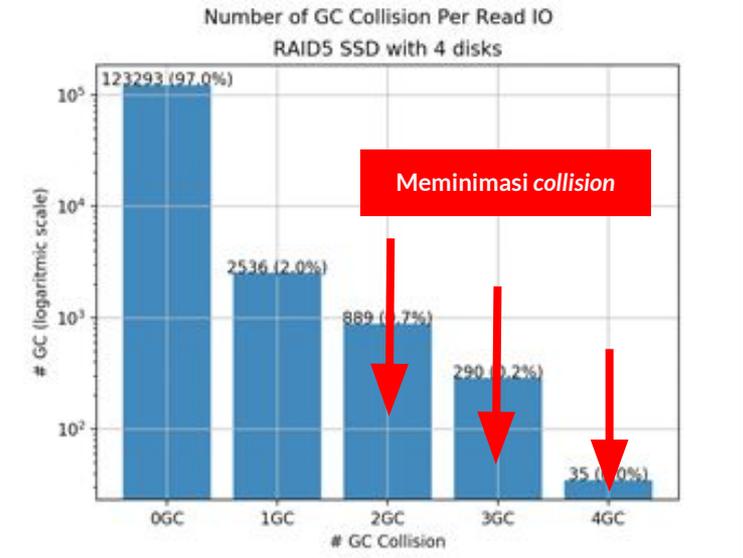
# Kondisi sebelum implementasi



**Masih banyak GC *collision* yang terjadi!**

Jika dilihat pada grafik CDF, jumlah GC dan GC collision sangat berkorelasi terhadap latensi di RAID SSD. Oleh karena itu penting untuk mengeliminasi dampak dari GC, langkah awalnya adalah dengan meminimasi *collision*.

# Kondisi yang diharapkan



- *GC Collision* dapat dikurangi jumlahnya, atau bahkan dihilangkan sama sekali.
- Harapannya 2GC, 3GC, 4GC, dst dapat diminimalisir, karena *collision* lebih dari satu GC tidak dapat diatasi dengan rekonstruksi paritas.
- 1GC dapat ditoleransi karena dapat diatasi dengan rekonstruksi menggunakan paritas.

# Implementasi Penjadwalan

## Modifikasi Simulator



Proses implementasi dan pengujian dilakukan pada, *trace-based simulator*, SSDSim yang dikembangkan oleh Yang Hu dan sudah diuji dengan *physical SSD* sungguhan.

Modifikasi dilakukan untuk mendukung simulasi RAID, terdapat komponen *controller* yang bertugas untuk memecah *request* ke SSD yang terhubung padanya. File-file statistik yang dihasilkan dari simulasi dapat diolah lebih lanjut untuk mengukur efektivitas implementasi.

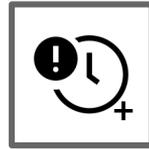
Source code simulator yang digunakan dapat diakses di <https://github.com/fadhilimamk/ssdsim>

# Rancangan Strategi Penjadwalan GC



**GCSync**

Menggunakan *time window* untuk mengatur GC pada RAID SSD. Proses GC pada SSD hanya dapat dilakukan pada saat *time window* tertentu.



**GCSync+**

Sama seperti GCSync, namun pada GCSync+ terdapat *buffer time* antar *time window*, sehingga dapat dipastikan tidak ada collision yang bisa terjadi.

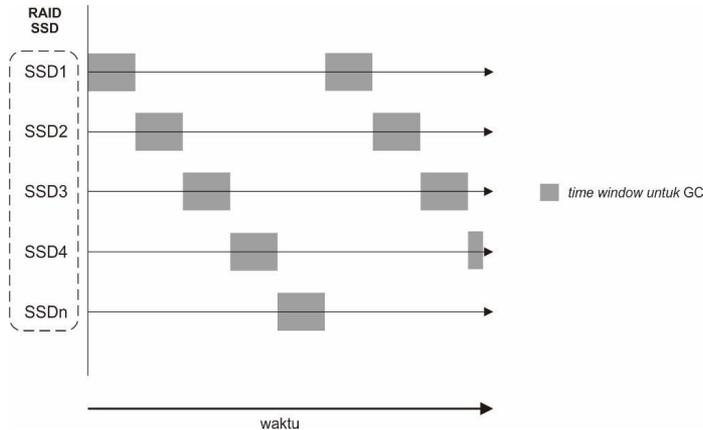


**GCLock**

Menggunakan lock pada *controller* RAID untuk mengatur GC pada RAID SSD. Proses GC pada SSD dapat dilakukan ketika *lock* sudah berhasil didapat dari *controller*.



# Implementasi Penjadwalan GCSync



Gambar ilustrasi *time window* pada GCSync untuk penjadwalan GC dalam RAID SSD

Metode GCSync memerlukan masukan *time\_window* dari pengguna.

Misalkan *time\_window* = 100ms, maka SSD dengan id  $c$  pada konfigurasi  $n$  disk memiliki *time window*:

$$[100(ni+c)]ms \text{ s/d } [100(ni+c+1)]ms, i \in Z$$



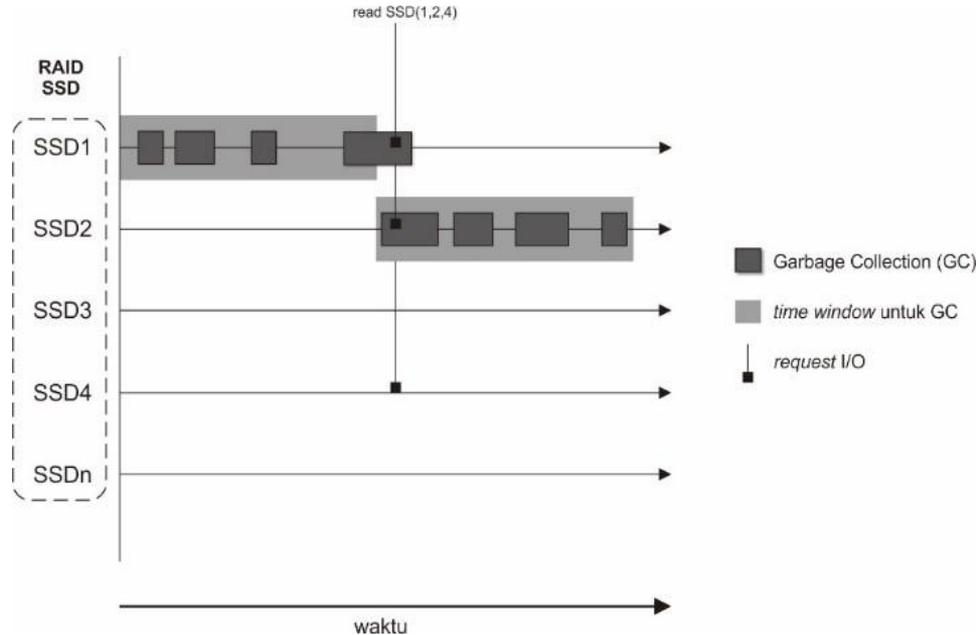
# Implementasi Penjadwalan GCSync

```
struct ssd_info{  
    ...  
    int is_gcsync;  
    int ndisk;  
    int diskid;  
    int64_t gc_time_window;  
    ...  
};
```

Modifikasi dilakukan pada *struct ssd\_info*,  
dengan tambahan atribut:

<code>is_gcsync</code>	Menandakan mode GCSync sedang aktif
<code>ndisk</code>	Jumlah disk pada konfigurasi RAID
<code>diskid</code>	ID dari disk yang bersangkutan, dimulai dari 0
<code>gc_time_window</code>	durasi <i>time_window</i> untuk GC

GCsync belum mampu menghilangkan semua *collision*.  
Masih ada kemungkinan 2GC terjadi, kenapa?



Hal ini berusaha dihindari pada **GCsync+** dengan menambahkan *buffer time* antar dua *time window*

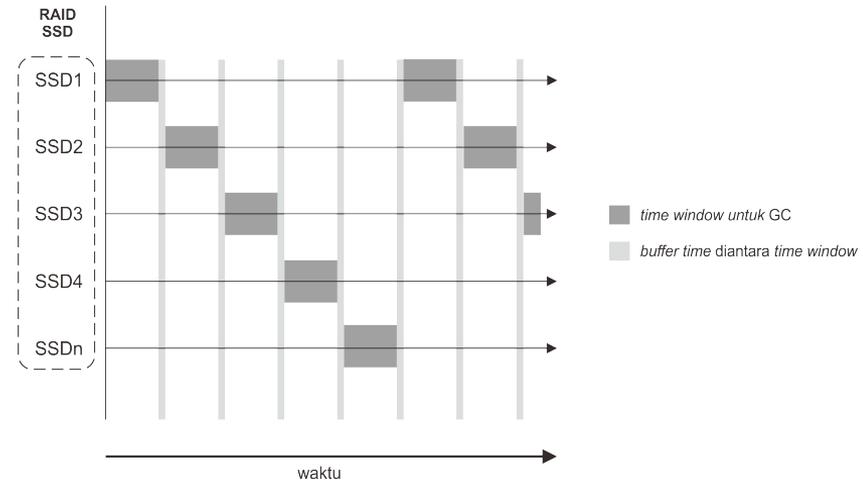


# Implementasi Penjadwalan GCSync+

Metode GCSync+ memerlukan masukan *time\_window* dan *buffer\_time* dari pengguna.

Misalkan *time\_window* = 100ms, maka SSD dengan id *c* pada konfigurasi *n* disk memiliki *time window*:

$$[(t_w + t_b)(ni + d_i)]ms \text{ s/d } [(t_w + t_b)(ni + d_i + 1) - t_b]ms, i \in Z$$



Gambar ilustrasi *time window* dan *buffer time* pada GCSync+ untuk penjadwalan GC dalam RAID SSD



# Implementasi Penjadwalan GCLock

Modifikasi dilakukan dengan menambahkan struct `gclock_raid_info` pada controller RAID, dengan atribut:

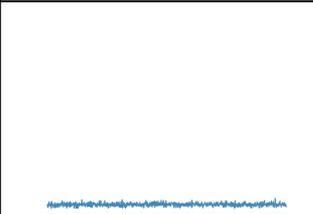
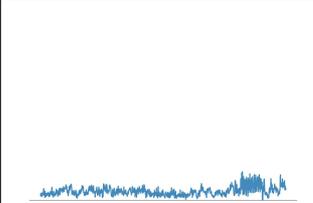
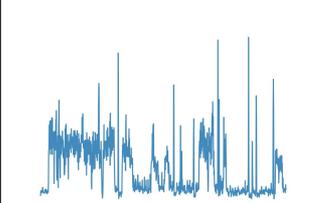
<code>begin_time</code>	Waktu mulai saat lock digunakan salah satu SSD
<code>end_time</code>	Perkiraan waktu proses GC selesai
<code>is_available</code>	TRUE jika dapat digunakan
<code>holder_id</code>	ID SSD yang sedang memegang <i>lock</i>

```
struct gclock_raid_info {  
    int64_t begin_time;  
    int64_t end_time;  
  
    int is_available;  
    int holder_id;  
};
```

Saat SSD akan melakukan GC, SSD tersebut perlu mengecek apakah *lock* tersedia di *controller*. Jika tidak tersedia maka proses GC digagalkan, dan menunggu SSD *men-trigger* proses GC selanjutnya.

# 7. Penguujian

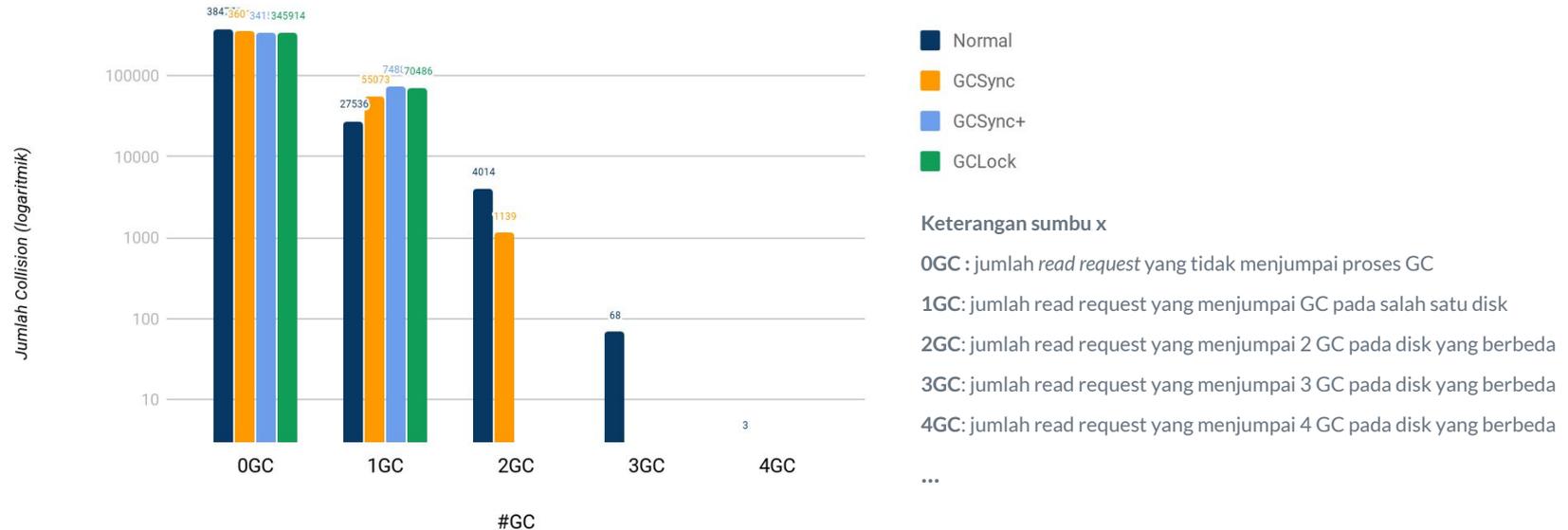
# Karakteristik *Workload* untuk Pengujian

<i>Workload</i>	Durasi (jam)	IO Rate	Avg. Req size Read/Write (KB)	Read (%)	Int. Arrv-Time (ms)
TPCC	0:50		8,57 / 7,99	36,16	7,20
DTRS	8:20		53,38 / 55,57	10,96	236,14
MSNFS	8:20		9,02 / 15,75	11,61	55,66

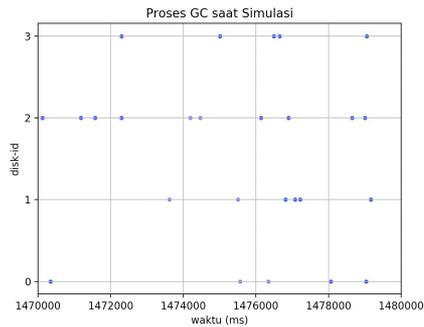
# Demo Simulasi

# Efektifitas Strategi Penjadwalan GC untuk Meminimasi *Collision* pada *Workload* TPCC

Jumlah GC Collision Sebelum dan Sesudah Implementasi Berbagai Strategi Penjadwalan GC

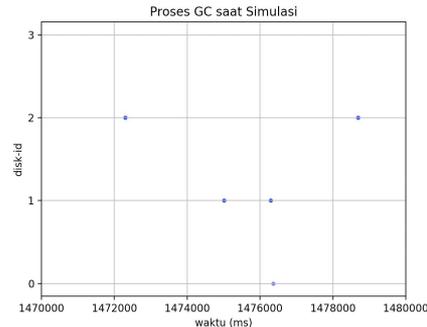


# Dampak Strategi Penjadwalan GC Terhadap Proses GC pada *Workload* TPCC



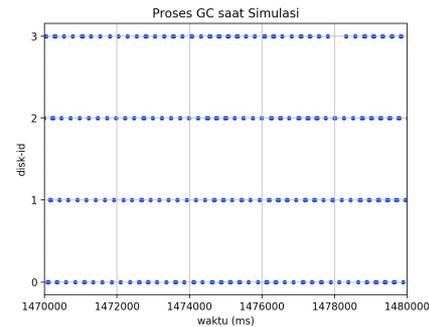
**Normal**

GC tidak beraturan karena tidak dikoordinasi



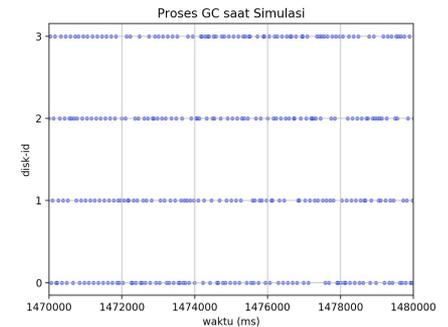
**GCSync**

GC lebih beraturan, *collision* sudah dapat dikurangi



**GCSync+**

GC lebih beraturan, *collision* dapat dihilangkan

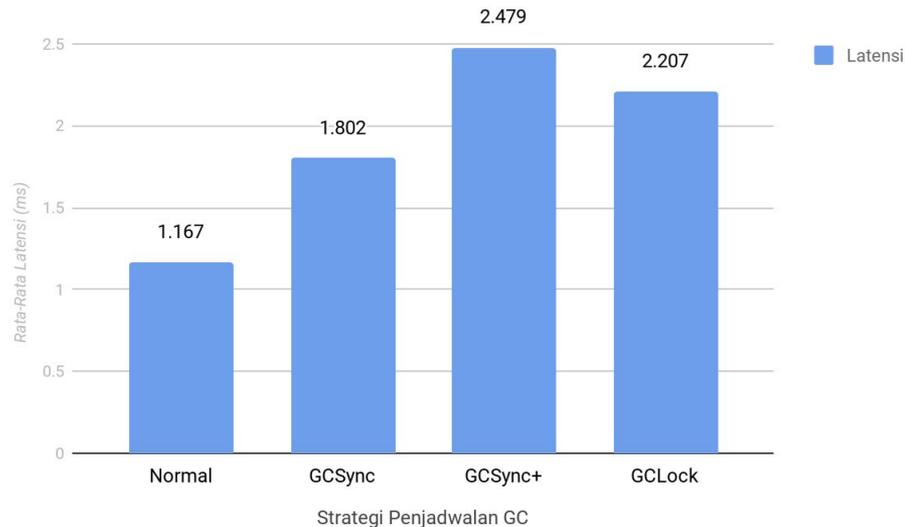


**GCLock**

GC lebih beraturan, *collision* dapat dihilangkan

# Pengaruh Strategi Penjadwalan GC terhadap Latensi *Read Request* pada *Workload* TPCC

Rata-Rata Latensi Sebelum dan Sesudah Implementasi Berbagai Strategi Penjadwalan GC



Implementasi strategi penjadwalan GC mengakibatkan kenaikan latensi hanya sebesar 1,54 - 2,13 x lipat

# Hasil Pengujian dengan *Workload* Lainnya

## Penurunan *GC Collision*

	Jumlah <i>Collision</i> (>2GC)			Meminimasi <i>collision</i> sebesar
	TPCC	DTRS	MSNFS	
Normal	3323	1214	3509	-
GCSync	1139	124	201	60 - 71%
GCSync+	0	0	0	100%
GCLock	0	0	0	100%

Strategi GCSync berhasil meminimasi *collision*, sedangkan GCSync+ dan GCLock menghilangkan keseluruhan *collision*

# Hasil Pengujian dengan *Workload* Lainnya

## Perubahan Latensi

	Rata-rata Latensi (ms)		
	TPCC	DTRS	MSNFS
Normal	1,167	0,415	0,854
GCSync	1,802	0,498	0,498
GCSync+	2,479	0,511	0,560
GCLock	2,207	0,519	0,547

Kenaikan latensi akibat implementasi strategi penjadwalan GC bervariasi antara 1,5 - 2,1 x lipat, namun pada workload MSNFS latensi justru turun, hal ini dipengaruhi oleh jumlah 1GC yang juga menurun.

8.

# Kesimpulan

# Kesimpulan

- Terdapat tiga strategi penjadwalan GC yang diusulkan untuk meminimasi GC *collision*, yaitu GCSync, GCSync+, dan GCLock.
- Strategi GCSync berhasil **mengurangi collision** sebesar 60-71% dan hanya memberikan kenaikan latensi antara 0,7-1,5 kali lipat.
- Strategi GCSync+ berhasil **menghilangkan seluruh collision** dan memberikan kenaikan latensi antara 0,7-2,1 kali lipat.
- Strategi GCLock juga berhasil **menghilangkan seluruh collision** pada RAID SSD, dan dari pengujian yang dilakukan, strategi ini memberikan kenaikan latensi sebesar 0,6-1,9 kali lipat.



## Saran

- Penelitian ini sudah berhasil meminimasi *GC collision* pada RAID SSD, untuk mengatasi 1GC dapat dilakukan penelitian terkait rekonstruksi menggunakan paritas pada RAID SSD.
- Penelitian dapat dilanjutkan dengan menambahkan lebih banyak *workload* dan mengukur pengaruh penjadwalan GC terhadap kinerja RAID SSD.
- Percobaan dengan simulator berbasis *virtual machine* dapat dilakukan untuk mengetahui kinerja RAID SSD dengan mempertimbangkan proses dalam OS (*Operating System*)

# Daftar Pustaka

- Aggrawal, Nitin dkk. (2008). *Design Tradeoffs for SSD Performace*. In Proceedings of the USENIX Annual Technical Conference (ATC), Boston, 2008.
- Card, S.K., Robertson, G.G., dan Mackinlay, J.D. (1991). *The information visualizer: An information workspace*. Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (New Orleans, Apr. 28– May 2). ACM Press, New York, 1991, 181–188.)
- Dean, Jeffrey dan Barroso, Luiz Andre. (2013). The Tail at Scale. Communications of the ACM, vol 56, pp. 74-80.
- Eshghi K., Micheloni R. (2018). SSD Architecture and PCI Express Interface. Dalam: Micheloni R., Marelli A., Eshghi K. (eds) Inside Solid State Drives (SSDs). Springer Series in Advanced Microelectronics, vol 37. Springer, Singapore.
- Hu, Yang dkk. (2011). *Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity*. Proceedings of the 25th International Conference on Supercomputing (ICS '11), Munich.
- Kim, Youngjae dkk. (2011). *Harmonia: A Globally Coordinated Garbage Collector for Arrays of Solid-state Drives*. 7th IEEE Symposium on Massive Storage Systems and Technologies and Co-located Events (MSST 2011), Denver.
- Lee, Sang-Won dan Kim, Jin-Soo. (2011). *Understanding SSDs with the OpenSSD Platform*. Seoul: Sungkyunkwan University.
- Li, Huaicheng dkk. (2018). *The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Simulator*. Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST '18), Oakland.
- Martindale, Jon. (2018). New laptops may see more storage as SSD prices expected to fall through 2019. <https://www.digitaltrends.com/computing/ssd-prices-fall-2019/>. Diakses pada 25 Oktober 2018.
- Miceloni, Roni. (2017). *Solid-State-Drives (SSD) Modeling: Simulation Tools & Strategies*. Singapore: Springer.
- Perumal, Sameshan dan Kritzinger, Pieter. (2004). *A Tutorial on RAID Storage Systems*. Cape Town: University of Cape Town.
- Skourtis, Dimitris dkk. (2013). *High Performance & Low Latency in Solid-State Drives Through Redundancy*. Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13), San Jose.
- Statista. (2017). Shipment of Hard and Solid-State Disk (HDD/SSD) Drives Worldwide from 2015 to 2021. <https://www.statista.com/statistics/285474/hdds-and-ssds-in-pcs-global-shipments-2012-2017/>. Diakses pada 30 September 2018.
- Swanson, Steven. (2011). Flash Memory Overview. San Diego: University of California San Diego.
- The OpenSSD Project. (2011). The Open SSD Project. [http://www.openssd-project.org/wiki/The\\_OpenSSD\\_Project](http://www.openssd-project.org/wiki/The_OpenSSD_Project). Diakses pada 21 Desember 2018.
- Yan, Shiqin dkk. (2017). *Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs*. Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST '17), Santa Clara.
- Yoo, Jinsoo dkk. (2013). *VSSIM: Virtual machine-based SSD simulator*. Proceedings of the 29th IEEE Symposium on Massive Storage Systems and Technologies (MSST' 13), Long Beach.

**Terima kasih!**