Available at https://jtsiskom.undip.ac.id (24 March 2020)
DOI:10.14710/jtsiskom.8.2.2020.157-163

Jurnal Teknologi dan Sistem Komputer, 8(2), 2020, 157-163

Strategi caching aplikasi berbasis in-memory menggunakan Redis server untuk mempercepat akses data relasional

Application caching strategy based on in-memory using Redis server to accelerate relational data access

Mulki Indana Zulfa[*)], Ari Fadli, Arief Wisnu Wardhana

Program Studi Teknik Elektro, Fakultas Teknik, Universitas Jenderal Soedirman
Jl. Mayjen Sungkono Km.5, Blater, Purbalingga, Indonesia 53371

Cara sitasi: M. I. Zulfa, A. Fadli, and A. W. Wardhana, "Strategi caching aplikasi berbasis in-memory menggunakan Redis server untuk mempercepat akses data relasional," Jurnal Teknologi dan Sistem Komputer, vol. 8, no. 2, pp. 157-163, 2020. doi: 10.14710/jtsiskom.8.2.2020.157-163, [Online].

Abstract – Utilization of an in-memory database as a cache can overcome relational database latency problems in a web application, especially when using a lot of join queries. This study aims to model the academic relational data into Redis compatible data and analyze the performance of join queries usage to accelerate access to relational data managed by RDBMS. This study used academic data to calculate student GPA that is modeled in the RDBMS and Redis in-memory database (IMDB). The use of Redis as an in-memory database can significantly increase Mysql database system performance up to 3.3 times faster to display student data using join query and to shorten the time needed to display GPA data to 52 microseconds from 61 milliseconds.

Keywords *– web application; relational data; in-memory database; Redis*

Abstrak – Pemanfaatan basis data in-memory sebagai cache dapat menjadi solusi untuk mengatasi latensi basis data relational dalam pengelolaan data terstruktur di aplikasi web, terutama di data relasional yang banyak menggunakan join query. Penelitian ini bertujuan mengkaji pemodelan data relasional akademik menjadi data yang kompatibel dengan Redis dan menganalisis kinerjanya dalam penggunaan join query untuk mempercepat akses data relasional yang dikelola oleh RDBMS. Kajian ini menggunakan data akademik untuk menghitung IPK mahasiswa yang dimodelkan dalam RDBMS dan in-memory database (IMDB) Redis. Penggunaan Redis sebagai basis data in-memory dapat menaikkan kinerja sistem Mysql secara signifikan hingga 1,7 kali lebih cepat dalam membantu mempersingkat waktu yang dibutuhkan dalam menampilkan data mahasiswa menggunakan join query dan waktu pembacaan data IPK hingga menjadi 52 mikrodetik dari 61 milidetik.

Kata kunci *– aplikasi web; data relasional; in-memory database; Redis*

[*)] Penulis korespondensi (Mulki Indana Zulfa)
Email: mulki_indanazulfa@unsoed.ac.id

I. PENDAHULUAN

Berkembangnya teknologi situs web berbanding lurus dengan jumlah pengguna layanannya. Cara mengelola data dalam aplikasi berbasis situs web harus dipikirkan karena setiap hari datanya terus bertambah serta dapat menyebabkan *bottleneck* dan Internet *latency* [1], [2]. Salah satu strategi untuk mengatasi hal tersebut adalah membuat sebuah sistem *caching* dari level basis data, aplikasi sampai jaringan [3].

Saat ini, aplikasi situs web masih banyak yang menggunakan *relational database management system* (RDBMS) sebagai media penyimpanan data utamanya. RDBMS ini mampu menyimpan data terstruktur dengan baik [4]. Data tersebut dapat direlasikan dari beberapa tabel sehingga dapat membentuk kesatuan informasi yang lebih lengkap. RDBMS juga memiliki kelebihan, yaitu jaminan terhadap sifat ACID yang harus terus dijaga selama melakukan transaksi di dalam basis data meliputi *atomicity*, *consistency*, *isolation*, dan *durability* [4], [5].

Namun, seiring dengan pertumbuhan data yang semakin cepat, maka data relasional ini akan semakin lambat untuk diakses. Sebagai contoh adalah aplikasi Sistem Informasi Akademik (SIA) di Universitas Jenderal Soedirman (Unsoed). Di tiap awal dan akhir semester, hampir semua mahasiswa mengakses data nilai akademik, terutama Indeks Prestasi Kumulatif (IPK). Bahkan, data ini digunakan sebagai proses evaluasi studi sehingga yang mengakses ini bukan hanya mahasiswa, namun juga para pengelola program studi [6]. Data IPK dibentuk dari relasi beberapa tabel menggunakan *join query*. *Join query* adalah sebuah skema menampilkan data yang disediakan oleh RDBMS dengan mengambil data dari beberapa tabel yang berbeda. Semakin banyak tabel yang dihubungkan, maka waktu eksekusi untuk *join query* akan semakin lama [7]. Permasalah tersebut disebabkan karena RDBMS menyimpan datanya di dalam *harddisk* [8].

Di sisi lain, teknologi penyimpanan data berbasis *in-memory database* (IMDB) sedang berkembang pesat. Teknologi ini banyak digunakan oleh jasa penyedia

Submitted: 24 June 2019; Revised: 18 March 2020; Accepted: 20 March 2020; Published: 30 April 2020

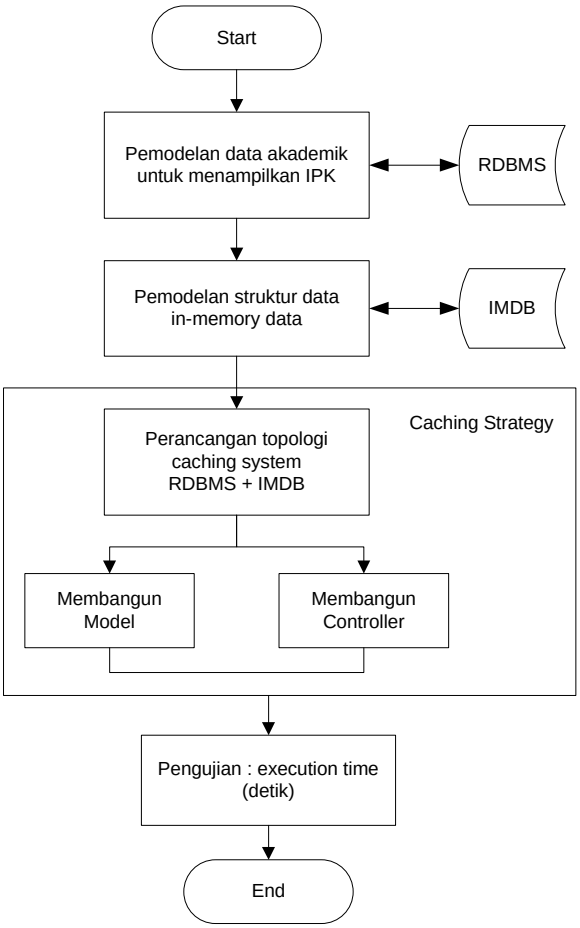
cloud, seperti Amazon Web Services, Google Cloud Platform, IBM, dan Microsoft Azure [9], [10]. IMDB tidak lagi menyimpan datanya di dalam *harddisk* tetapi disimpan di dalam memori komputer sehingga memiliki kecepatan akses yang jauh lebih baik [8].

IMDB menggunakan konsep penyimpanan *key-value* dan tidak lagi membutuhkan *structured query language* (SQL) sehingga dikenal sebagai basis data NoSQL [11], [12]. Dari hal ini, IMDB sangat berbeda dengan RDBMS untuk mengakses data yang berada di dalamnya. NoSQL didasarkan pada teori CAP, yaitu pemilihan dua dari tiga aspek yang ada yang harus dipenuhi oleh basis data, yaitu *consistency*, *avaibility*, dan *partition-tolerance* [4], [13]. Contohnya adalah aspek *consistency-avaibility* (CA) akan berseberangan dengan aspek *partition-tolerance* sebagai aspek paling penting dalam replikasi [14]. IMDB dapat menjadi basis data pendamping dengan kecepatan akses yang lebih baik daripada RDBMS. Selain itu, IMDB juga dapat mengurangi beban kerja RDBMS sebagai basis data utama aplikasi. Dari hal tersebut, IMDB dan RDBMS dapat digunakan bersama untuk saling melengkapi, khususnya dalam strategi *caching* untuk mempercepat akses data relasional.

Beragam perangkat IMDB telah banyak digunakan. Evaluasi perbandingan kinerja dari IMDB dengan Memcached, H2, Redis, Cassandra, dan MongoDB telah dilakukan dalam [15]. Evaluasi diukur berdasarkan waktu eksekusi dan memori yang digunakan dan menyatakan bahwa Redis paling efektif dalam penggunaan memori, sedangkan Cassandra dan Redis adalah yang tercepat dalam menampilkan data. Selain itu, Redis juga paling cepat dalam menghapus data.

Implementasi *HTTP Reserve Proxy* Varnish dan Redis sebagai *cache* basis data dilakukan dalam [16]. *Cache* ini digunakan untuk meringankan beban kerja server web dengan cara menyimpan sementara konten web pada memori agar dapat digunakan kembali pada *request* yang sama. Kombinasi Varnish dan Redis dapat mempercepat akses hingga 15-16 kali dibandingkan dengan server web yang tidak menggunakan *cache*. Lebih lanjut, hasil ini diujicobakan pada Wordpress dengan basis data Mysql dalam [17]. Penggunaan Redis dapat mengurangi beban kerja server basis data hingga 25 % dan mengurangi secara signifikan aktivitas pembacaan *harddisk* hingga 60 %.

Kajian ini bertujuan untuk memodelkan data relasional akademik menjadi data yang kompatibel dengan Redis dan menganalisis kinerja Redis, khususnya dalam penggunaan *join query* sehingga dapat mempercepat akses data relasional yang dikelola oleh RDBMS. Redis sebagai IMDB diimplementasikan untuk aplikasi berbasis web. Berbeda dengan [15]-[17] yang menggunakan aplikasi Wordpress, kajian ini mengimplementasikan Redis untuk aplikasi SIA yang dikembangkan dengan Codeigniter serta menjabarkan jenis dan dan struktur datanya. Selain itu, kajian ini melakukan perancangan struktur data Redis yang digunakan sehingga kompatibel dengan struktur data relasional.



Gambar 1. Alur penelitian yang dilakukan

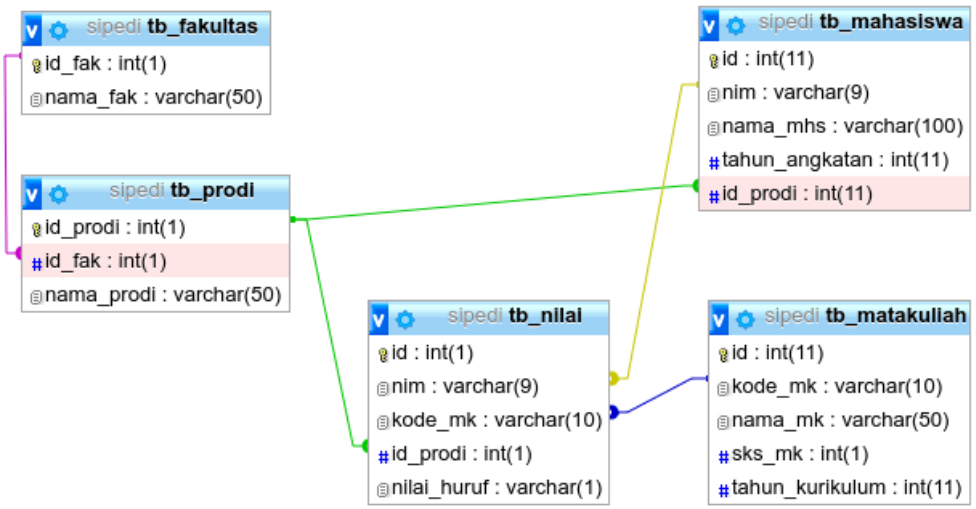
II. METODE PENELITIAN

Alur penelitian ini dinyatakan dalam Gambar 1. Penelitian dimulai dengan membangun data akademik pada basis data Mysql (RDBMS) untuk mensimulasikan proses perhitungan IPK mahasiswa. Struktur data IMDB dibuat agar dapat mengakomodasi data akademik karena sudut pandang IMDB tidak lagi menggunakan tabel, namun menggunakan konsep penyimpanan *key-value.* Perangkat IMDB yang dipilih adalah Redis.

Setelah data akademik dapat dimodelkan RDBMS dan IMDB, langkah berikutnya adalah merancang *Model* dan *Controller* menggunakan Codeigniter untuk mengintegrasikan keduanya dalam aplikasi berbasis web. Aplikasi ini diuji untuk menampilkan data IPK dengan dua skenario, yaitu ketika *request* data sudah berada di dalam *cache* dan ketika data tersebut belum berada di dalam *cache*. Variabel yang diukur adalah waktu respon, dalam detik, yang didapatkan pengguna ketika meminta data IPK.

A. Pemodelan data akademik

Data akademik dimodelkan ke dalam beberapa tabel. Hal ini ditujukan agar data IPK yang ditampilkan diambil dari beberapa tabel yang saling mempunyai relasi. Untuk memodelkan data akademik yang dapat menampilkan data IPK, diperlukan lima tabel, yaitu



Gambar 2. Relasi data akademik untuk menampilkan IPK mahasiswa

tabel mahasiswa, tabel fakultas, tabel prodi, tabel matakuliah, dan tabel nilai. Relasi dari lima tabel tersebut dapat dilihat pada Gambar 2. Kelima tabel ini diakses menggunakan *join query* untuk menampilkan data IPK mahasiswa. Berdasarkan struktur pada tabel-tabel tersebut, struktur data pada IMDB dibuat agar dapat mengakomodasi setiap nilainya.

B. Pemodelan struktur data IMDB

Redis sebagai perangkat *opensource* berlisensi BSD dapat menyimpan struktur data dalam memori. Redis dapat menjadi sebuah IMDB, *cache* atau *broker message*. Redis ini mendukung beberapa struktur data, seperti *string, hashes, lists, sets, sorted sets* hingga data *geospasial*, dan *bitmaps* [18]. Redis dan IMDB lain, seperti memcached dan Ehcache [19], dapat berkolaborasi dengan beberapa bahasa pemrograman, seperti C/C++, PHP, Java, dan Python [20]-[22] dan telah teruji berjalan dengan baik pada sistem operasi Linux [23], [24]. Oleh karena itu, penelitian ini menggunakan Redis yang dipasang pada Linux turunan Ubuntu 16.04, yaitu Linux LiteOS 4. Pustaka PHP-Redis yang digunakan adalah *Predis* [11].

Operasi dasar pada Redis yang digunakan adalah `get` dan `set`. Operasi `set` membutuhkan sebuah *key* untuk menyimpan data. Data ini dapat diubah dan dihapus. Setiap *key-value* dapat diberikan waktu kadaluarsanya (*expired*) menggunakan operasi `setex`. Operasi ini dapat menentukan *time-to-live* (TTL) atau lamanya masa tinggal sebuah data di dalam *cache* [25].

Tipe data yang paling sering digunakan oleh Redis adalah *string*. Redis membatasi ukuran string, yaitu 512 Megabytes. Redis juga mempunyai tipe data *hash* yang dapat menyimpan banyak *key* di dalam suatu *key*. Operasi terhadap tipe data ini dapat menggunakan HMSET [18]. Dalam penelitian ini, struktur data yang digunakan oleh Redis disesuaikan dengan struktur tabel data akademik yang telah tersimpan pada basis data Mysql. Namun, karena Redis tidak mengenal relasi data, maka dalam penelitian ini, data akademik tersebut dibuat dengan *key* yang menggunakan Nomor Induk Mahasiswa (NIM). Semua data mahasiswa dimuat dalam *key* ini. Penelitian ini menggunakan data mahasiswa Teknik Elektro dari angkatan tahun 2000 sampai dengan angkatan 2015.

C. Perancangan model dan controller

Untuk menampilkan data IPK mahasiswa, dibutuhkan proses perhitungan bobot nilai matakuliah dikalikan dengan sks dan dibagi dengan total sks sehingga proses perhitungan IPK merelasikan setidaknya dua tabel. Tetapi dalam implementasinya, data IPK ini ditampilkan bersamaan dengan data identitas akademik lainnya, seperti NIM, Nama Mahasiswa, Nama Matakuliah, Tahun Angkatan, Nama Prodi dan Nama Fakultas sehingga semua data ini saling berelasi.

Perancangan model

Dalam *framework* Codeigniter, model digunakan untuk membangun koneksi dengan basis data sekaligus mendefinisikan *query* yang digunakan untuk mengambil data dari basis data. Contohnya adalah *join query* yang digunakan untuk menampilkan data semua mahasiswa seperti dapat dilihat pada Algoritme 1. Contoh *join query* yang digunakan untuk menampilkan data IPK mahasiswa berdasarkan NIM dapat dilihat pada Algoritme 2.

Perancangan controller

Dalam *framework* Codeigniter, *controller* digunakan untuk mengatur permintaan data yang berasal dari pengguna melalui *browser* dan diteruskan kepada model yang bertanggungjawab untuk mengelola data yang berkaitan dengan permintaan pengguna tersebut. Dalam penelitian ini, *controller* memiliki peranan penting untuk mengatur aliran data yang masuk dan keluar dari Redis.

Controller berfungsi melakukan pemetaan struktur data Mysql ke struktur data Redis karena Redis

Algoritme 1. Sintaks *query* untuk menampilkan data mahasiswa

```
select
a.nim as nim, a.nama_mhs as nama, a.tahun_angkatan as
tahun, b.nama_prodi as prodi
from tb_mahasiswa a
join tb_prodi b on a.id_prodi = b.id_prodi
```

Algoritme 2. Sintaks *query* untuk menampilkan data IPK

```
select
a.nim, b.nama_mhs,
round(sum((c.sks_mk * a.bobot))/sum(c.sks_mk),2) as
totalnilai
from tb_nilai a
join tb_mahasiswa b on a.nim = b.nim
join tb_matakuliah c on c.kode_mk = a.kode_mk
where a.nim = 'C1X015070'
GROUP by a.nim
```

memiliki struktur data yang berbeda dengan Mysql. Penelitian ini menggunakan struktur data *hash* yang mampu menyimpan banyak *key* di dalam suatu *key*. Operasi terhadap tipe data ini menggunakan HMSET. Strateginya adalah ketika ada permintaan data mahasiswa atau IPK, maka *controller* memanggil model dengan menggunakan *query* pada Algoritme 1 dan Algoritme 2 untuk menampilkan data semua mahasiswa dan menampilkan data IPK sesuai dengan NIM mahasiswa. Alur strategi caching data ditunjukkan dalam Gambar 3.

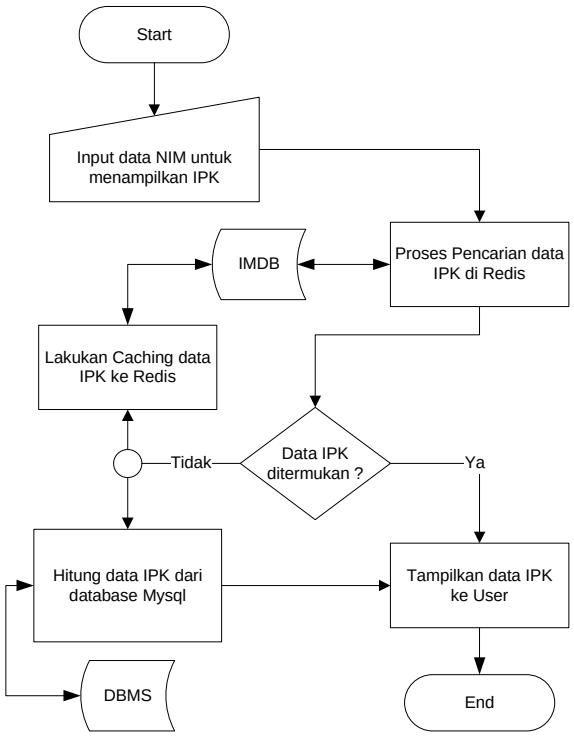
Hasil perhitungan IPK tersebut disimpan ke dalam cache Redis agar ketika ada pengguna lain yang meminta data yang sama, maka permintaan ini tidak lagi diteruskan ke Mysql melainkan dijawab langsung oleh Redis. Strategi ini dapat meringankan beban server basis data sehingga akan mempercepat akses menampilkan data IPK yang merupakan data relasional yang dibentuk dari beberapa tabel. Cara yang sama digunakan untuk mekanisme *caching* data mahasiswa. Jika data mahasiswa sudah ada di dalam Redis, maka setiap permintaan data yang sama dapat langsung diberikan kepada pengguna tanpa harus melakukan *query join* ke Mysql. Namun jika data mahasiswa belum terdapat dalam *cache*, maka aplikasi harus memintanya terlebih dahulu kepada Mysql.

III. HASIL DAN PEMBAHASAN

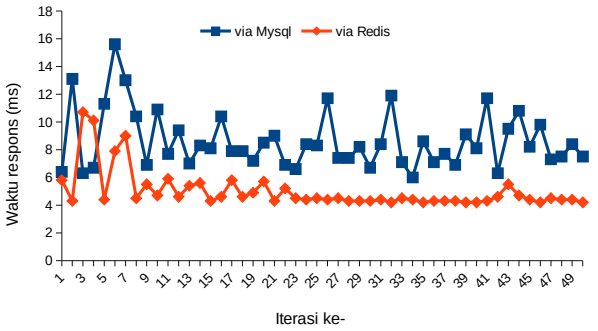
Strategi *caching* ini diuji coba untuk mengukur kecepatan waktu respon (*execution time*) berdasarkan tiga skenario. Skenario 1 dilakukan untuk membandingkan waktu eksekusi yang dibutuhkan untuk menampilkan semua data mahasiswa, baik dari basis data Mysql maupun dari Redis. Skenario 2 dilakukan untuk membandingkan waktu eksekusi dalam menampilkan data IPK, baik dari basis data Mysql maupun dari Redis. Skenario 3 dilakukan untuk melihat waktu yang dibutuhkan untuk menulis data ke dalam Redis.

A. Hasil skenario 1

Skenario 1 membandingkan waktu eksekusi yang dibutuhkan Mysql maupun Redis untuk menampilkan



Gambar 3. Alur strategi *caching* data



Gambar 4. Perbandingan waktu respons menampilkan data semua mahasiswa

semua data mahasiswa. Data mahasiswa yang ditampilkan sudah dalam format lengkap meliputi nama lengkap, NIM, nama prodi, dan fakultas. Data mahasiswa diambil dari tiga tabel dan ditampilkan dengan menggunakan *join query* seperti yang dinyatakan pada Algoritme 1.

Berdasarkan hasil pengujian pada Gambar 4, waktu eksekusi yang dibutuhkan Redis dalam menampilkan data semua mahasiswa lebih cepat daripada waktu yang dibutuhkan basis data Mysql. Rata-rata waktu eksekusi yang dibutuhkan oleh Mysql dalam menampilkan semua data mahasiswa adalah 8,63 milidetik dari 50 iterasi percobaan, sedangkan rata-rata waktu eksekusi yang dibutuhkan oleh Redis adalah 5,02 milidetik.

Namun, ada hasil yang berbeda pada iterasi percobaan ke-3 dan ke-4. Waktu yang dibutuhkan Redis untuk menampilkan semua data mahasiswa ternyata lebih lambat daripada waktu yang dibutuhkan oleh Mysql (via DB). Salah satu penyebabnya adalah di waktu iterasi tersebut berjalan, penggunaan memori

komputer sedang mengalami interupsi sehingga alokasi ruang memori yang digunakan oleh Redis ikut terganggu dan menyebabkan waktu akses menjadi lebih lambat.

Kondisi tersebut memang sebuah anomali karena jika diperhatikan secara keseluruhan, maka pola grafik yang dihasilkan antara waktu respon Redis dan Mysql hampir selalu sama. Hal ini berarti bahwa jika komputer sedang mengalami interupsi pada memori, maka besar kemungkinan hal yang sama juga terjadi pada *harddisk*. Namun, di iterasi ke-3 dan ke-4 terlihat bahwa ketika memori mengalami interupsi justru di saat yang sama tidak terjadi interupsi pada *harddisk* sehingga waktu akses data dari *harddisk* melalui Mysql lebih cepat dari memori (Redis).

B. Hasil skenario 2

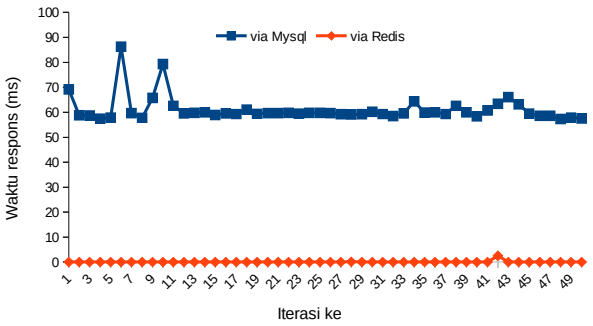
Skenario 2 membandingkan waktu eksekusi yang dibutuhkan Mysql maupun Redis untuk menampilkan data IPK mahasiswa berdasarkan NIM. Data IPK adalah data relasional yang diambil dari dua tabel, yaitu tabel matakuliah dan tabel nilai. Namun, data yang ditampilkan bukan hanya IPK, tetapi lengkap dengan data mahasiswa lainnya sehingga tabel yang saling mempunyai relasi bukan hanya 2, tetapi 5 tabel. Query yang digunakan dalam skenario ini dinyatakan pada Algoritme 2.

Berdasarkan hasil pengujian menampilkan data IPK mahasiswa pada Gambar 5, waktu eksekusi yang dibutuhkan Redis dalam menampilkan data IPK ratusan kali lebih cepat daripada waktu eksekusi yang dibutuhkan Mysql. Namun, perbedaan waktu yang sangat signifikan ini bukanlah hasil kinerja murni dari Redis. Tugas Redis sebenarnya adalah Redis hanya menampilkannya saja tanpa melakukan proses perhitungan data IPK di Mysql sehingga waktu yang dibutuhkan Redis untuk menampilkan data IPK sangat cepat sekali.

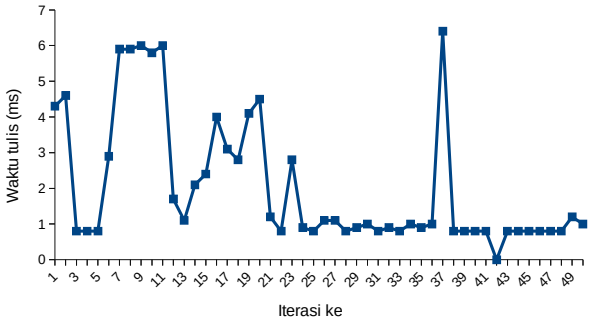
Di antara iterasi 3 s.d 11 terlihat pola grafik yang cukup tinggi pada garis biru (akses basis data via Mysql), namun pola ini tidak diikuti oleh garis oranye (akses data via Redis). Hal ini disebabkan pada skenario 2 ini data yang ditampilkan oleh Redis hanya satu data IPK mahasiswa saja sehingga penggunaan memori juga sangat ringan sekali dan tidak berpengaruh ketika ada interupsi yang ada. Hal ini menjadi strategi yang sangat baik dengan memanfaatkan Redis sebagai *cache* untuk menampilkan data relasional, khususnya yang melibatkan proses perhitungan dari hasil *join query* yang cukup membebani proses komputasi. Jika hasil dari komputasi yang cukup berat ini dapat disimpan di dalam cache, maka ketika ada permintaan data IPK dari mahasiswa yang sama, aplikasi tidak lagi melakukan proses perhitungan ulang Mysql, melainkan cukup menampilkan data yang sudah berada di dalam *cache* Redis.

C. Hasil skenario 3

Berdasarkan ilustrasi strategi *caching* pada Gambar 3, maka penelitian ini juga telah mencatat waktu yang dibutuhkan untuk menulis data IPK ke dalam *cache* Redis. Skenario ketiga ini melakukan pencarian data



Gambar 5. Perbandingan waktu respons menampilkan data IPK mahasiswa



Gambar 6. Waktu yang dibutuhkan untuk menulis *cache* data dalam Redis

IPK berdasarkan NIM mahasiswa. Pencarian data IPK ini langsung dilakukan dari Redis dengan harapan permintaan data IPK mahasiswa dapat direspon dengan cepat. Namun, jika data yang dicari tidak ditemukan di dalam Redis maka pencarian data IPK dilanjutkan ke Mysql. Mysql menghitung nilai IPK mahasiswa dan hasilnya langsung ditulis ke dalam *cache* Redis menggunakan perintah `zadd('key', value1, value2, value-ke-N)`. `Key` adalah nama kunci unik yang diingat oleh Redis jika ada permintaan suatu data, sedangkan `value` adalah nilai – nilai yang dapat disimpan di dalam Redis yang melekat terdapat suatu `key`.

Mekanisme ini adalah salah satu strategi *caching* untuk permintaan data IPK mahasiswa sehingga jika ada permintaan data IPK yang sama, maka data tersebut langsung dapat direspon oleh Redis dengan waktu yang lebih cepat. Berdasarkan Gambar 6, waktu rata-rata yang dibutuhkan Redis untuk menulis data IPK ke dalam *cache* dalam 50 kali percobaan adalah 2 milidetik, dengan waktu paling cepat adalah 0,1 mikrodetik dan waktu paling lambat mencapai 6,4 milidetik.

Redis memanfaatkan ruang memori yang terbatas dan harus berbagi dengan sumber daya lainnya ketika ada yang membutuhkan ruang memori untuk proses komputasi. Walaupun hanya menulis data IPK mahasiswa dengan kolom-kolom yang sama, namun waktu yang dibutuhkan dapat berbeda-beda karena hampir selalu terjadi interupsi di dalam memori tergantung beban kerja server saat itu. Ketika beban kerja server sedang tinggi karena harus melayani beban komputasi lainnya, maka waktu yang dibutuhkan untuk

menulis data di dalam *cache* menjadi cukup lama. Hal ini ditunjukkan dalam Gambar 6 pada iterasi 1, 2, 6-11, 14-20, dan 23. Puncaknya adalah iterasi ke-37, Redis membutuhkan waktu menulis data di dalam *cache* paling lama dalam percobaan ini adalah 6,4 milidetik.

D. Pembahasan

Hasil percobaan pada skenario 1 menunjukkan bahwa kecepatan akses data melalui Redis jauh lebih cepat daripada kecepatan akses data melalui basis data (Mysql) hingga 1,72 kali dari 8,63 milidetik menjadi 5,02 milidetik. Pada skenario 2, ketika terjadi proses perhitungan IPK mahasiswa yang cukup kompleks, keberadaan Redis sebagai basis data pendamping sangat tepat. Bahkan, hasil skenario 2 menunjukkan bahwa kecepatan Redis dalam menampilkan data IPK lebih cepat daripada Mysql, yaitu waktu respons rata-ratanya berkurang hingga menjadi 0,1 % dari 61 milidetik menjadi 52 mikrodetik. Kecepatan baca dan tulis suatu data ke dalam Redis menjadi jauh lebih cepat karena semua proses yang terjadi di dalam Redis dilakukan melalui memori. Di sisi lain, waktu baca dan tulis data ke dalam Mysql jauh lebih lambat karena semua proses yang terjadi di dalam Mysql dilakukan pada *harddisk* dengan *latency* yang lebih besar sehingga waktu yang dibutuhkan untuk akses data menjadi jauh lebih lama.

Hasil skenario 1 dan 2 ini menunjukkan kesesuaian hasil penelitian dengan teori yang dikemukakan dalam [15], [16] yang menyatakan bahwa data yang disimpan di dalam memori lebih cepat diakses daripada data yang tersimpan di dalam *harddisk*. Redis yang memanfaatkan memori sebagai media penyimpanan data [12], juga memiliki resiko karena pada dasarnya semua proses komputasi membutuhkan sumber daya dari memori [10]. Kinerja Redis juga dipengaruhi oleh interupsi, permintaan alokasi memori secara tiba-tiba, yang terjadi di dalam memori [22]. Jika pada saat yang bersamaan proses interupsi sering terjadi, maka performa Redis juga ikut turun, seperti ditunjukkan pada Gambar 4 iterasi percobaan ke-3 dan ke-4.

Namun, di balik resiko tersebut, Redis sangat membantu kinerja server basis data sebagai basis data pendamping [23]. Data yang tersimpan di Redis dapat lebih cepat diakses. Kecepatan akses ini menyebabkan waktu tunggu (*delay*) yang dialami pengguna menjadi berkurang. Begitupun dengan waktu respons yang dibutuhkan server dalam menampilkan hasil *join query* yang jauh lebih cepat jika dibandingkan harus ditampilkan tanpa bantuan Redis.

Hal tersebut dapat dilihat pada Gambar 5. Semua data yang dilayani dari Redis memiliki waktu proses 60x lebih cepat daripada jika dilayani tanpa melalui Redis. Hal ini membuktikan Redis secara efektif membantu meringankan beban kerja server seperti dalam [17] karena permintaan data yang sama tidak harus dimuat ulang dari server. Hasilnya sudah ada di dalam *cache* Redis. Redis membantu sebagai media yang dapat mempercepat akses data relasional dari hasil *join query* yang membutuhkan komputasi besar [7]. Beban kerja server yang lebih ringan pada akhirnya menimbulkan efisiensi energi pada sumber daya server yang dikelola [4].

Efisiensi energi lainnya juga dapat dilihat pada hasil pengujian skenario 3 (Gambar 6). Jika saja Redis membutuhkan waktu yang lama untuk menulis data, maka akan ada banyak proses komputasi lain yang terhambat karena harus menunggu alokasi ruang memori kembali bebas dan siap untuk menerima komputasi proses selanjutnya sesuai [25].

IV. KESIMPULAN

Redis mampu mempercepat akses data relational dalam aplikasi SIA dibandingkan melalui basis data Mysql secara langsung yang menggunakan *join query*. Selain itu, Redis ini menghasilkan rata-rata waktu untuk menuliskan data ke cache Redis sebesar 2 milidetik dari 50 percobaan. Dari hal tersebut, Redis secara efektif dapat meringankan beban kerja server basis data. Performa Redis sangat bergantung pada beban komputasi yang dialami oleh server sehingga untuk penelitian berikutnya perlu dirancang secara khusus mekanisme adaptif yang dapat menyeimbangkan beban kerja server sehingga kecepatan akses data melalui Redis tetap terjaga.

UCAPAN TERIMA KASIH

Penelitian ini dibiayai oleh Kemenristek Dikti melalui LPPM Universitas Jenderal Soedirman yang telah membiayai penelitian ini dalam skim Penelitian Peningkatan Kompetensi.

DAFTAR PUSTAKA

[1] J. Yan, J. Chen, and W. Jiang, "Data caching techniques in web application," in 2014 Enterprise Systems Conference, Shanghai, Chine, Aug. 2014, pp. 289-293. doi: 10.1109/ES.2014.57

[2] S. Bouchenak, A. Cox, S. Dropsho, S. Mittal, and W. Zwaenepoel, "Caching dynamic web content: designing and analysing an aspect-oriented solution," in *ACM/IFIP/USENIX 7th International Middleware Conference*, Melbourne, Australia, Dec. 2006, pp. 1-21. doi: 10.1007/11925071_1

[3] W. Ali, S. M. Shamsuddin, and A. S. Ismail, "A survey of web caching and prefetching," *International Journal of Advances in Soft Computing and its Applications*, vol. 3, no. 1, pp. 1-27, 2011.

[4] M. Indrawan-santiago, "Database research : are we at a crossroad ? reflection on NoSQL," in *15th International Conference on Network-Based Information Systems*, Melbourne, Australia, Sept. 2012, pp. 45-51. doi: 10.1109/NBiS.2012.95

[5] A. E. Lotfy, A. I. Saleh, H. A. El-Ghareeb, and H. A. Ali, "A middle layer solution to support ACID properties for NoSQL databases," *Journal of King Saud University - Computer and Information*

Sciences, vol. 28, no. 1, pp. 133-145, 2016. doi: 10.1016/j.jksuci.2015.05.003

[6] A. Fadli, M. I. Zulfa, and Y. Ramadhani, "Perbandingan unjuk kerja algoritme klasifikasi data mining dalam sistem peringatan dini ketepatan waktu studi mahasiswa," *Jurnal Teknologi dan Sistem Komputer*, vol. 6, no. October, pp. 158-163, 2018. doi: 10.14710/jtsiskom.6.4.2018.158-163

[7] J. Sinuraya, "Metode pencarian data menggunakan query hash join dan query nested join," *Teknovasi*, vol. 4, no. 1, pp. 42-50, 2017.

[8] M. Luthfi, M. Data, and W. Yahya, "Perbandingan performa reverse proxy caching nginx dan varnish pada web server apache," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 2, no. 4, pp. 1457-1463, 2018.

[9] R. Nishtala et al., "Scaling memcache at facebook," in *10th USENIX conference on Networked Systems Design and Implementation*, Chicago, USA, Apr. 2013, pp. 385-398.

[10] C. Liu, K. Ouyang, X. Chu, H. Liu, and Y. Leung, "R-Memcached : a reliable in-memory cache for big key-value stores," *Tsinghua Science and Technology*, vol. 20, no. 6, pp. 560-573, 2015. doi: 10.1109/TST.2015.7349928

[11] W. Puangsaijai and S. Puntheeranurak, "A comparative study of relational database and key-value database for big data applications," in *2017 International Electrical Engineering Congress*, Pattaya, Thailand, Mar. 2017, pp. 8-10. doi: 10.1109/IEECON.2017.8075813

[12] D. J. Carlson, *Ebook redis in action*. Manning Publications, 2013.

[13] N. L. Seth Gilbert, "Brewer's conjecture and the feasibility of consistent, available, and partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51-59, 2002. doi: 10.1145/564585.564601

[14] F. Firdausillah, E. Y. Hidayat, and I. N. Dewi, "NoSQL : latar belakang, konsep, dan kritik," in *Seminar Nasional Teknologi Informasi dan Komunikasi Terapan*, Semarang, Indonesia, Jun. 2012, pp. 432-438.

[15] A. T. Kabakus and R. Kara, "A performance evaluation of in-memory databases," *Journal of King Saud University - Computer and Information Sciences*, vol. 29, no. 4, pp. 520-525, 2017. doi: 10.1016/j.jksuci.2016.06.007

[16] M. Kusuma, "Evaluasi performa web server menggunakan varnish http reserve proxy dan redis database cache," in *Seminar Nasional Inovasi dan Aplikasi Teknologi Industri*, Malang, Indonesia, Feb. 2016, pp. 260-264.

[17] M. Kusuma, W. Widyawan, and R. Ferdiana, "Performance comparison of caching strategy on wordpress multisite," in *3rd International Conference on Science and Technology - Computer*, Yogyakarta, Indonesia, Jul 2017, pp. 176-181. doi: 10.1109/ICSTC.2017.8011874

[18] Redislabs, "Redis," 2014. [Online]. Available: https://redis.io/commands. [Accessed: 23-Nov-2018].

[19] A. P. Negrão, C. Roque, P. Ferreira, and L. Veiga, "An adaptive semantics-aware replacement algorithm for web caching," *Journal of Internet Services and Applications*, vol. 6, no. 4, pp. 1-14, 2015. doi: 10.1186/s13174-015-0018-4

[20] H. Zhang, G. Chen, and C. Ooi, "In-memory big data management and processing : a survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920-1948, 2015. doi: 10.1109/TKDE.2015.2427795

[21] D. Li, M. Dong, Y. Yuan, J. Chen, K. Ota, and Y. Tang, "SEER-MCache : a prefetchable memory object caching system for iot real-time data processing," *IEEE Internet of Things Journal*, vol. PP, no. 8, pp. 3648-3660, 2018. doi: 10.1109/JIOT.2018.2868334

[22] Y. Fan, Y. Wang, and M. Ye, "An improved small file storage strategy in ceph file system," in *14th International Conference on Computational Intelligence and Security*, 2018, pp. 488-491. doi: 10.1109/CIS2018.2018.00116

[23] F. Pedone and D. L. Epfl, "A primary-backup protocol for in-memory database replication," in *IEEE International Symposium on Network Computing and Applications*, Cambridge, USA, Jul. 2006, pp. 204-211. doi: 10.1109/NCA.2006.7

[24] F. Pedone, "Sprint : a middleware for high-performance transaction," in *ACM SIGOPS Operationg System Review*, vol. 41, no. 3, pp. 385-398, 2007. doi: 10.1145/1272998.1273036

[25] K. Dutta and D. Vandermeer, "Caching to reduce mobile app energy consumption," *ACM Transactions on the Web*, vol. 12, no. 1, pp. 1-30, 2017. doi: 10.1145/3125778