School of Informatics

Informatics Project Proposal
ML-based data prefetching in PostgreSQL

B175601
January 2022

Abstract

Data prefetching for a database management system is proactively fetching data from disk to memory before the point of data reference[1]. It can hide observable latencies by fully utilizing CPU, increasing the buffer hit ratio and thus improve the system performance. In many database systems, implementing data prefetching is proved to be possible and beneficial especially due to the characteristic of access sequentiality[2]. Our project intends to implement a ML-based data prefetcher in the PostgreSQL database system for speeding up queries.

Date: Monday 24th January, 2022

Tutor: Vijay Nagarajan
Supervisor: Milos Nikolic

1 Motivation

Due to large data volume, disk is often the secondary storage for a database system. Each time an operation occurs, it may require data that is not currently in the main memory buffer, thus disk I/O is evoked to fetch data from the disk to the memory. Such is called fetching data on demand. Obvious latency would be observed when the system can not continue with the current operation but, normally, wait for the I/O to complete till the data is available in the buffer. Sometimes, the system would divert to other jobs that are not blocked. Nevertheless, the performance is still undermined due to additional operations.

The performance of the system under this scenario can be abstracted and reflected with the buffer hit ratio. To improve the system performance would be to increase the buffer hit ratio. The easiest way to increase the ratio is to increase the buffer size, which theoretically can be enormous and contain the complete database. However, we are only to discuss a practical case where we have memory buffer that is limited and has fixed length. The other way would be to guarantee that each time a page of data is required, it is already in the buffer. Therefore, it will always be a hit and the thread wouldn't need to wait for I/O.

This is why data prefetching is proposed. For a page of data to be already in the buffer when the system demands for it, either the page was requested before on demand and hasn't been replaced for storing other pages in the buffer, or the page was not in the buffer but the system somehow know that the page would soon be requested and thus has fetched it from the disk before the actual demand. The latter case is what I refer to as data prefetching. There are two key elements that are critical for data prefetching to actually work well and they are 1) the location to fetch and 2) the timing[1].

Which location is to be fetched? To fetch a page of data from the disk, we need to know its identifier, which we can simplify as pageID. In a typical operation of sequential scan, we would request consecutive records of data in a run. It would be obvious that after requiring record(i), record($i+1$), we might be needing record($i+2$), etc. So long as the records are stored in a sequential way in pages of data, we could deduce that after getting page(j), page($j+1$) might have the records that will be requested later. Therefore, under such assumption, we have candidate pages to be prefetched. This assumption can be clearly stated as that the logical sequentiality of access is reflected in physically sequential access to the stored data[2]. It is a characteristic of many systems called sequentiality. The database management system we study here — PostgreSQL shares such characteristic.

When shall we prefetch? The timing is always associated with the degree of prefetching and that is how many pages to prefetch. If prefetching is too late, access latency won't be hidden. If prefetching comes too early and the page will remain idle for a long time, the buffer hit ratio is restricted. The page that is the victim for replacing the buffer slot might try to find its way back to the memory buffer and cause page replacement again, which involves frequent and, sometimes, useless I/O. Same can be said when the degree of prefetching is not moderate.

Among common prefetching methods in database management systems, hardly any has combined the implementation of machine learning techniques. They commonly derive heuristics from observation to decide when to prefetch. The degree of prefetching is also fixed in accordance with certain features of the history of page requests, such as learning from the distribution of reference sequence length. Since that machine learning can sometimes capture patterns that are not explicit to human and might adjust prefetching techniques according to the history and not just to one feature extracted, maybe we can implement a ML-based data prefetcher in PostgreSQL that can do smarter prefetching and improve system performance.

1.1 Problem Statement

In a nutshell, the problem behind this project is a question —

- Would a ML-based data prefetcher in PostgreSQL improve the system performance?

To answer that question, I would need to

- Implement a ML-based data prefetcher in PostgreSQL.
- Make reasonable comparisons between the system with the aforementioned prefetcher and the system without prefetching or with naive prefetching.

1.2 Research Hypothesis and Objectives

The necessary hypothesis would be —

- Only the effects of prefetching is considered. It is a critical assumption, especially after knowing that prefetching and caching are intertwined[3]. It is easy to imagine that the page replacement policy in the buffer makes great difference to the performance of the system. However, we keep the policy unchanged (Clock policy in PostgreSQL) and only consider the effects of prefetching under such policy.
- PostgreSQL shares the characteristic of sequentiality. Otherwise, prefetching would be almost impossible and useless.
- The system would be studied on a specific system on the same benchmark. Especially in training ML-based prefetcher, the only way to determine parameters would be driven by the same benchmark and consistent observed behaviour.
- For making performance comparison, only use results measured against the same database size. Since that the TPC believes that comparisons of TPC-H results measured against different database sizes are misleading and discourages such comparisons[4].

To achieve the ultimate goal of implementing a ML-based data prefetcher in PostgreSQL, there are several research objectives.

- Collect page requests under different workloads. It requires setting a benchmark with large database and different workloads. I would observe the behaviour of the buffer manager of PostgreSQL and collect page requests as traces for later use.
- Learn few predictive models for temporal and spatial data prefetching based on the traces. I need to select appropriate machine learning models such as Long Short Term Memory networks(LSTM)[5] that can capture sequential patterns.
- Add functionality into PostgreSQL. It requires understanding the real code of PostgreSQL and integrate the learned model into the system.
- Experimentally evaluate the models. Using appropriate metrics to measure the performance of the system with or without the functionality. Some conventional prefetching techniques may be used as baselines.

1.3 Timeliness and Novelty

Data prefetching is relatively harder than instruction prefetching. Therefore, it could be said that data prefetching is less developed. Data prefetching in database management systems are even less discussed in comparison with more explicit methods such as page replacement policy. Among the methods already used for data prefetching optimization in database management systems, they are generally less adaptive but fixed according to statistics and human observation. No machine learning techniques have been used to directly develop a data prefetcher for database management systems from my observation.

In the current world of big data, the optimization of commercialised database management systems would be important for removing any obstacles in processing large volume of data. Combining machine learning with different aspects of computer architecture is nowadays on trend. Our project would be a first bold trial in this certain aspect of marrying database management systems and machine learning.

1.4 Significance

This project could be a first in implementing a data prefetcher using machine learning method. It may not give a better result for the PostgreSQL, but it extends the range of exploration in every way. First, it widens the possibility of using machine learning techniques. Second, it adds a new approach for optimizing data prefetching. Third, whatever the result may be, the outcome would be enlightening. For example, if the ML-based prefetcher doesn't work well, it still presents an attempt of such method and reasonable analysis could be given on why is this the case. If it turns out beneficial, it would be an improvement on the already very-well developed database management system. And its success could be populated to other database management systems and improve the system performance to the next level. After the project, other research on this field could be explored like using different machine learning methods or combining it with more background information.

1.5 Feasibility

To complete the project, there are multiple steps and they are somewhat in a blocked way. Therefore, it's important to have a compact schedule in the early stage and try to achieve concurrency. We can generate sufficient traces using TPC-H[4] for machine learning method to train on. The traces are in statistics and not in larger chunks like images, therefore the time for training would be less. I would use the cluster resource from the university, which would give even faster results, thus allowing multiple trials. The first prototype of ML-based data prefetcher in PostgreSQL should be trained early on so that enough time would be given to the integration (into PostgreSQL system), performance analysis and modifications. Baseline models for comparison can be developed during the training period.

1.6 Beneficiaries

Other researchers can learn from my project a new way of using machine learning as a tool, explore relevant prefetcher and enrich the field of data prefetching in database management system. In industrial context, the success can be populated to other database management systems. The improvement of the performance of the system will benefit data workers and subsequently the big data industries.

2 Background and Related Work

In order to hide I/O latency, caching is used fundamentally. It has been used in different aspects of computer architecture, for example in storage controllers, file systems, operating systems, etc.[3] Paging mechanism for virtual memory management is a typical example of caching in OS. To further increase the buffer hit ratio, two main approaches are used[6]. By selecting the better replacement policy, the page soon be needed can be kept in the buffer pool and the page won't be used in the long term will be kicked out. By using prefetching, the page soon to be needed will be fetched into the buffer pool. These two strategies compensate each other and are usually discussed together. For example, to optimise cache utilization, [7] created a predictive cache system with an ordered list of pages with expected access time.

Setting replacement policy aside, cache prefetching is used for fetching data or instructions that are soon to be needed[8]. In this way, the two separated mechanisms — I/O and memory can work concurrently thus the speed of the computer processor can be accelerated. Sine that the access patterns of instructions are more explicit than that of data, data prefetching is believed to be harder than instruction prefetching[9]. There are two ways of implementing cache prefetching — using hardware or software[10]. The prefetching in our discussion would be software prefetching because we are to hardcode what is learnt by the machine learning model into the higher-level code of PostgreSQL. In this way, the prefetcher is inserted into the program and is fixed during compilation[11].

[2] gives the most relevant work on database buffer prefetching. It proposed a notion of reduced block referenced string (RBRS) which is used to preprocess the traces generated from page requests. The main focus is on the run length and its distribution which would give preference to which degree of prefetching would be given the current run length. [1] is one of the rare works that considered combining machine learning technique with data prefetching. However, the K-means unsupervised clustering methods used to partition data into clusters is more like auxiliary tool which helps detect the preferred prefetching configurations based on workload patterns. [12] combines prefetching with machine learning. It introduced the notion of semantic locality and developed a context-based memory prefetcher using reinforcement learning.

Different methods of prefetching in database have been explored. [13] proposed to utilize context information on relational databases. The context information could be anything like a stored collection of relationships. And when an object is fetched, other objects with similar states would be prefetched. [14] proposed using data compression techniques to preform prefetching. Such prefetching is based on Lempel-Ziv data compressor and [15] gave practical analysis on using data compression techniques for prefetching. [16] proposed a new prefetching policy called Selective Eager Object Fetch (SEOF), which is designed for object-oriented database systems and uses no high-level object semantics.

It is important to know about the mechanism of the buffer manager in PostgreSQL[17]. For a request (backend process), it carries the unique identifier (buffer tag) of the page needed. This buffer tag is composed of various information that are sufficient for recognizing where the page is on the disk. For example, it contains which block in which relation contained in which database under which tablespace the page is in by storing the OID of these objects. In the most complicated case, the request evokes loading a page from disk to a victim buffer pool slot. It means that the desired page is not in the buffer pool and the buffer pool is full. Therefore, we need to choose a victim of page to be replaced by our desired page using a page replacement policy, which in PostgreSQL is Clock-Sweep algorithm. For each page in the buffer pool, it has a mapping (stored in the buffer table) from the buffer tag to the buffer id of the slot it is in.

3 Programme and Methodology with Evaluation

The project starts with data generation. We refer to the data here as traces that will be used for training machine learning model. Traces are the observed behaviour of buffer manager under sequences of page requests. Therefore, in order to generate traces, we need to generate the database and queries first. The database is supposed to be the playground for the prefetcher. Different set of queries would be classified as different workloads.

The playground database and queries would be generated using the TPC-H benchmark. TPC-H provides a base database with fixed number of tables and fixed common features that is dynamic (refreshed frequently) and has high industry-wide relevance. By using the population generator — DBGEN and selecting a desired scale factor, I can easily scale up the database and provide large volumes of data for experiment. For generating queries, I can use QGEN to translate the given 22 kinds of query templates into the executable query text. Each query can be customised based on the template and is of high complexity and representativity.

Before generating traces using the aforementioned database and queries, I need to know what information of the behaviour of the buffer manager shall be included inside the traces and how to collect them. To test its feasibility, I can start with naive workloads that simply evokes sequential scan. I am to find tools to help me observe what is going on inside the buffer manager of PostgreSQL and use code to automatically transform information into traces of useful format. After learning about some of the details of the buffer manager of PostgreSQL mentioned in section 2, I can deduce that after collecting sequence of pages requested, the information in the buffer tag of each page can be used. For example, the relation OID and block OID. Due to the characteristic of sequentiality, these OID would be critical in deciding the future pages to be fetched.

I can now automatically generate traces using the database and queries. Traces are sequences of the information of the pages requested. Now, several choices might be taken into consideration. For example, would any context information be needed for training? To put it simply, whether a choice is considered depends on what kind of ML-based data prefetcher I am after.

Now, I am to narrow the concept of our prefetcher. Our ML-based data prefetcher shall be distinguished from ordinary ones. The base PostgreSQL system is considered to be the original system without prefetching or with a naive prefetching policy. Such a naive prefetching policy could be — given a page request on page(j), the system would always fetch page(j) and prefetch page($j+1$) (if not in the buffer pool). The location to prefetch is relatively fixed for every page request. The timing of prefetching is also fixed because the system evokes prefetching every time there is a page request. This naive prefetching policy may be too naive and can be further modified to be less stupid. For example, evoke prefetching on the next page every time the current page requesting is not in the buffer pool. The degree of prefetching, that is how many pages to prefetch, can also be changed.

Therefore, our prefetcher will also take those modifiable factors into account and they are 1) which page(s) to prefetch, 2) the degree of prefetching and 3) when to prefetch. The first factor is better left to the prefetcher to decide. The third factor could be fixed or decided by the prefetcher. It would be simpler if the timing is fixed.

If the timing is fixed, the prefetcher would need to decide the n next pages to be prefetched. The reason why we need to have a moderate degree of prefetching is to avoid memory pollution, which means fetching blocks that are never used. In light of this additional cost, we might train the machine learning model to always keep a low memory pollution. Therefore, the information for calculating this Bad-Fetch Cost (BFC) should be collected for training.

If the timing is flexible, we can transform the factors into one parameter and that is the possibility of prefetching that page. A maximum number of pages that can be prefetched shall be set. Within such range, prefetcher can decide what pages may be referenced later and each page would be assigned a possibility of prefetching that indicate a combination of the degree of prefetching and the timing. It's tricky as to how to feedback to the model the calculation of the possibility. The status of the buffer pool may be needed. The hit ratio, cost and even sequence length distribution mentioned in [2] that are ad hoc may be calculated. Notably, it may be better to keep the traces not reduced for better pattern recognition. These details shall be decided according to actual progress and analysis.

With the needed traces and, possibly, additional information, I shall train several machine learning models. The actual machine learning techniques to be explored and chosen according to the training feature of the data provided. Different models will be trained and compared. It would be impossible to compare models without integrating them into the system. Therefore, each model shall be integrated into the system and get ready to be compared with base systems.

The metrics used would be adopted from TPC-H. The performance of a prefetcher would be reflected with the difference in the performance of the system. Therefore, the performance of each system will be evaluated using the performance test of the benchmark, which consists of a power test and a throughput test[4]. The metric can be the TPC-H composite query-per-hour performance metric or simply the total query execution time. There are other considerations in the testing scenario. For example, shall we use the same database or would scaling the database be sufficient for changing the database into a different one from the playground for prefetcher? In order to make fair metric comparison, all systems shall be measured under the same database, which is the particular case of TPC-H shall not be databases with different scale factor. Moreover, ordinary test scenario may be compared with test under disk pressure simulation according to [18] since that the effects of prefetching can be more obvious under disk pressure. Furthermore, the graph of current disk queue length can be useful for reflecting the performance. In the end, an average case analysis for random workloads would be given.

There are certain limitations implied here in the project. The data used for training and testing are simulated and derives from the same source. Although working on a typical benchmark can be enough for experiments, it may be better to test performance on other benchmark as well. The analysis of system performance is less fine-grained. The effects of prefetching is averaged out and it would be hard to detect the mechanism of the prefetcher in individual operations.

3.1 Risk Assessment

Risk Name	Mitigation
Data generation takes too long.	Ask supervisor and email its students who have experience.
Traces not sufficient for training.	Scale up database and run queries several times.
Prefetcher too complicated to train.	Simplify the function of prefetcher by fixing certain features.
Model doesn't work well.	Train several models. Compare and give analysis.

Table 1: Risk assessment of the project.

3.2 Ethics

All the data used are generated and do not suffer from ethics issues.

4 Expected Outcomes

The main outcome would be an usable ML-based data prefetcher in PostgreSQL. The prefetcher is expected to be smarter and thus improves the system performance. In the end, an overall evaluation of PostgreSQL systems with different prefetching features will be given.

The project would fill the empty field of using machine learning techniques for data prefetching in PostgreSQL. It would delve further into the details of the buffer manager of PostgreSQL and give enlightening examples of using machine learning techniques for prefetching in database management systems.

5 Research Plan, Milestones and Deliverables

The schedule is set compact in the early period of the project which gives tolerance to unexpected long hour of any work package.

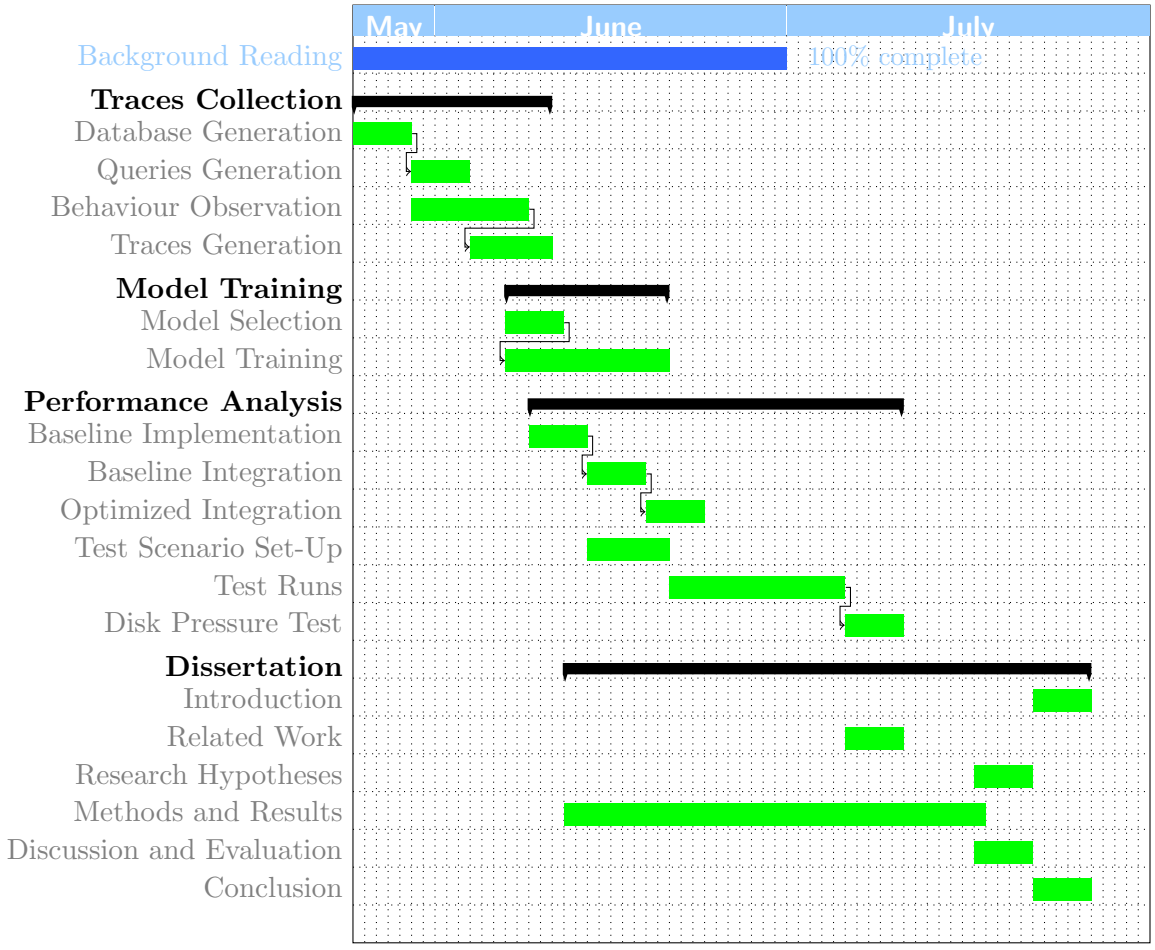

Figure 1: Gantt Chart of the activities defined for this project.

Milestone	Week	Description
M_1	3	Traces collection completed
M_2	4	First model training completed
M_3	4	Baseline integrated system completed
M_4	5	First optimized integrated system completed
M_5	8	Test runs completed
M_6	11	First draft of dissertation
M_7	13	Submission of dissertation

Table 2: Milestones defined in this project.

Deliverable	Week	Description
D_1	2	Generated Database and queries
D_2	3	Collection of traces
D_3	4	A first prefetcher model trained
D_4	4	Baseline integrated system
D_5	5	A first optimised integrated system
D_6	9	Dissertation: methods and results summary
D_7	11	First draft of dissertation
D_8	13	Final Dissertation

Table 3: List of deliverables defined in this project.

References

[1] Diana Guttman, Mahmut Taylan Kandemir, Meena Arunachalam, and Rahul Khanna. Machine learning techniques for improved data prefetching. In *5th International Conference on Energy Aware Computing Systems and Applications, ICEAC 2015*. Institute of Electrical and Electronics Engineers Inc., 12 2015.

[2] Alan Jay Smith. Sequentiality and Prefetching in Database Systems. Technical report.

[3] Binny S Gill and Dharmendra S Modha. SARC: Sequential Prefetching in Adaptive Replacement Cache. Technical report.

[4] TPC BENCHMARK TM H. Technical report, 1993.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[6] Steven P. Vanderwiel and David J. Lilja. Data Prefetch Mechanisms. *ACM Computing Surveys*, 32(2):174–199, 6 2000.

[7] Predictive cache system. Technical report, 8 1994.

[8] Prefetching - Wikipedia.

[9] Cache prefetching - Wikipedia.

[10] Y Solihin. *Fundamentals of parallel multicore architecture*. 2015.

[11] AK Porterfield. Software methods for improvement of cache performance on supercomputer applications. 1989.

[12] L Peled, S Mannor, U Weiser 2015 ACM/IEEE 42nd . . . , and Undefined 2015. Semantic locality and context-based prefetching using reinforcement learning. *ieeexplore.ieee.org*.

[13] Philip A. Bernstein, Pal Shankar, and David Shutt. Context-based prefetch - An optimization for implementing objects on relations. *VLDB Journal*, 9(3):177–189, 2000.

[14] Jeffrey Scott Vitter and P. Krishnan. Optimal prefetching via data compression, 1996.

[15] Kenneth M. Curewitz, P. Krishnan, and Jeffrey Scott Vitter. Practical Prefetching via Data Compression. *ACM SIGMOD Record*, 22(2):257–266, 1 1993.

[16] Jung Ho Ahn and Hyoung Joo Kim. SEOF: An adaptable object prefetch policy for object-oriented database systems. In *Proceedings - International Conference on Data Engineering*, pages 4–13. IEEE, 1997.

[17] The Internals of PostgreSQL : Chapter 8 Buffer Manager.

[18] SQL Server Prefetch and Query Performance - Simple Talk.