Milestone 2 *

Subtitle Text, if any †

Param Manhas ‡
SFU 5th Year Student
paramm@sfu.ca

Harwinder Dhillon
SFU 4th Year Student
hda26@sfu.ca

Mubanga Ben Ngosa §
SFU 4th Year Student
mngosa@sfu.ca

Abstract

PEREGRINE is a pattern-aware multi threaded graph mining system that runs on a single machine system. It breaks up the data mining program into 2 sections, Pattern selection and Pattern Matching. Users describe what patterns should be explored, then mining is started by matching patterns starting from each vertex and calling user functions on the pattern to process. These are the main tasks that the matching threads handle. These matching tasks are operated and distributed on dynamically to threads using a global atomic counter that leads to the next vertex to start the matching process on. Furthermore the PEREGRINE system does on-the-fly aggregation while it mines, which is handled by a an asynchronous aggregator thread that maintains the global aggregate values. This can be used for early termination and other cases. Thanks to this design, the matching threads remain non-blocking maintaining consistent high throughput. Studies done by K. Jamshidi, R. Mahadasa. and K. Vora have shown that the PEREGRINE system scales well with the addition of more physical cores and shows potential to scale well with a distributed system implementation. This proposal goes over the multi threaded logic used in PEREGRINE and discusses possible approaches of how scaling it across multiple machines would work and the potential issues associated with the approach.

CCS Concepts • **Software and its engineering** → **General programming languages**; • **Theory of computation** → *Program analysis*

* with optional title note

† with optional subtitle note

‡ with optional author note

§ with optional author note

[Copyright notice will appear here once 'preprint' option is removed.]

Keywords

1. Introduction

PEREGRINE is an efficient pattern-aware multi-threaded graph mining system that runs on a single machine. Other state-of-the-art graph mining systems don't take into account the patterns that they are mining for to help narrow down the search space. This limits other mining systems in a couple of ways; these systems perform a large amount of unnecessary computation in processing and verifying every sub graph that is explored, on graphs that are very large. The exhaustive searching method they use is very memory intensive and can be very costly, guided exploration strategies can help reduce the memory cost but not as well as PEREGRINE with its pattern-aware guided exploration. PEREGRINE breaks up the data mining into 2 sections: Pattern Selection and Pattern Matching. Users describe what patterns should be explored, with the ability to specify, vertices, edges, size,labels, and more. They can directly operate on patterns and create callback functions to perform analysis on the returned sub graphs.

2. Multithreading in PEREGRINE

In PEREGRINE, multithreading works like this:

1. Independent Tasks: PEREGRINE processes data in a way where each task doesn't depend on others. Explorations starting from two different vertices do not require any coordination meaning different tasks can start from different points without needing to wait for each other. Threads dynamically pick up new tasks when they finish their current ones.

2. Stopping Exploration: For certain types of tasks, like finding something specific pattern in the graph, threads can actively monitor conditions. When one thread finds what it's looking for, it can tell the others to stop looking.

3. Mining Patterns: PEREGRINE looks for patterns in the graph by starting from different points and using a user-defined function to process those patterns.

4. Aggregation for Progress: PEREGRINE keeps track of progress by periodically combining results from different threads. This helps in making decisions like when to stop

searching, especially when some patterns are found to be frequent. An ”aggregator” thread collects and combines these results from others, and once it's done, it signals that the final result is ready.

5. Implementation in C++: PEREGRINE is built using C++. It uses multiple threads that work on different parts of the data graph. These threads don't need to constantly check with each other to know what to do next. Instead, they use a shared counter to keep track of where to start their work, minimizing the need for coordination.

6. Load Balancing: Since workers in PEREGRINE pick up new tasks as soon as they're available, the workload is evenly distributed. This means that all threads finish their tasks around the same time, resulting in very little difference in the time it takes for each worker to complete its work.

In simpler terms, PEREGRINE works as team of workers exploring a maze. Each worker can explore different parts of the maze on their own without waiting for instructions. When one worker finds what they're looking for, they can shout to the others to stop exploring, and everyone eventually shares their findings to make decisions. This teamwork is efficient, and everyone finishes their work at almost the same time, making the workload very balanced.

3. Possible Approach Scaling PEREGRINE

Studies done by K. Jamshidi, R. Mahadasa. and K. Vora have shown that the PEREGRINE system scales well with the addition of more physical cores and shows potential to scale well with a distributed system implementation. Currently PEREGRINE's multi-threading logic works mostly asynchronously. The tasks are independent of each other since explorations of vertices don't need coordination. The matching threads which are responsible for the pattern matching and processing make up majority of the threads. Aside from the match finding, they have the responsibility of monitoring required conditions, which if seen needs to be communicated to other threads to invoke the shutdown process. To minimize communication and synchronicity, matching threads maintain data about their exploration tasks. The main important thing that is shared with all the threads is a shared atomic counter that points to the next vertex to be worked on. There is another thread that is an aggregator thread that periodically performs aggregation s local aggregate values appear from the matching threads. It remains blocked until all the matching threads have sent their aggregate values. This means the matching threads will not block and retain high throughput. It was noticed in the referenced paper that PEREGRINE threads was observed to have a near-zero load balance. There does not appear to be a large need for synchronicity between the threads.

Depending on the resources and machines we are scaling this mining program to, We believe that a decentralized asynchronous architecture will be easier to scale and match close to the speedup as the scaling of physical cores of the single machine program. As with asynchronous distributed systems, the challenges will come from potential complexity and inconsistency. MPI is a potential library we could use for thread communication, in sending the local aggregate values and more. We would need to account for failures in communications from checking if a node has died or is just really slow at sending a message would be a concern or partially send/received messages. As our system would be asynchronous, we would look into using Paxos to help solve(mitigate) the consensus problem.

A. Appendix Title

This is the text of the appendix, if you need one.

Acknowledgments

Acknowledgments, if needed.

References

K. Jamshidi, R. Mahadasa. and K. Vora. ...Peregrine: A Pattern-Aware Graph Mining System...