Fast String Searching on PISA

Theo Jepsen[1,2], Daniel Alvarez[2], Nate Foster[2,3], Changhoon Kim[2], Jeongkeun Lee[2], Masoud Moshref[2], and Robert Soulé[1,2]

[1] Università della Svizzera italiana
[2] Barefoot Networks
[3] Cornell University

String searching is one of the most common and important functions that run on computers. It is estimated that 80% of the world's data is unstructured, meaning that it cannot be easily queried using a fixed data model. Instead, users must search through large amounts of log data, JSON files, email, web pages and other documents to find the relevant patterns. These searches can have a significant impact on application performance. For example, Palkar et al.[4] recently showed that using string searches to pre-filter data before feeding it into Spark can provide a 9× improvement in end-to-end application completion time.

Unfortunately, data is increasingly stored on devices that cannot provide good search performance. In order to improve the utilization of resources, many data centers have turned towards a disaggregated architecture, in which storage devices have weak CPUs and little memory. In the storage community, these devices are commonly referred to as JBODS—just a bunch of disks.

Besides the fact that these machines are "wimpy", performing search also requires paying the levitation cost to get data off the storage. When running search on an x86 CPU, this will be through PCI-Express, which is a known bottleneck.

In fact, we argue that the general approach of having an x86 CPU inspect every single byte of incoming data is fundamentally flawed. With bulk data for storage or big data applications, the main mode of operation from the x86's point of view should be DMA. In other words, the CPU should only handle metadata or parts of the data required for real computing (e.g., mapping or transforming).

Given that I/O is a bottleneck and data is distributed on a large number of machines, we believe there is an exciting opportunity to leverage the emerging trend of repurposing high-speed programmable networking ASICs for non-networking workloads: accelerating string search. After all, networking interconnects can easily reach high throughput performance of multi-Tbps. Moreover, in purely economic terms, using a dedicated ASIC for search would be less expensive than increasing the specifications of storage servers.

We present PISA-based Parallel Search (PPS)[5]‘, a system for locating occurrences of string keywords stored in the payload of packets. The intuition behind PPS is to store DFA state transitions in the match tables of the pipeline. However, to achieve high-performance, PPS relies on two key techniques. First, PPS partitions the DFA into a set of smaller DFAs that run in each stage, allowing PPS to benefit from the pipeline parallelism inherent in the ASIC design. This also allows PPS to use the relatively small (compared to DRAM) on-chip SRAM more efficiently. Second, in order to increase throughput, PPS uses switch SRAM and TCAM to support larger DFA *stride* sizes (i.e., the number of characters matched per transition). Beyond these key techniques, PPS provides an optimization that trades-off accuracy for memory usage by matching on the CRC16 hash of the input characters, rather than the characters themselves.

We have implemented a prototype of PPS, which includes a compiler, a custom network controller, and a small client application to send data to the network. We have also incorporated our prototype into Apache Spark, allowing PPS to accelerate basic data analytics. Our evaluation shows that PPS demonstrates significantly higher throughput and lower latency than string searches running on CPUs, GPUs, or FPGAs.

In this demo, we will use PPS to search various files: JSON data, log files and unstructured text. PPS has a command line client like `grep`, so we can search any file on the filesystem, without any special preparation. We will stream the files to a switch running PPS, which supports a search throughput of 3.8 Tbps for 100 search strings. Because it can sustain this high throughput, PPS can perform string search on behalf of a cluster of commodity servers, freeing-up their CPUs for other work.

[4] Shoumik Palkar, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Filter Before You Parse: Faster Analytics on Raw Data with Sparser. VLDB (July 2018)

[5] To appear in SOSR'19