A Parallel and Pipelined Architecture for Accelerating Fingerprint Computation

Dongyang Li, Qingbo Wang, Cyril Guyot, Ashwin Narasimha, Dejan Vucinic, Zvonimir Bandic, Qing Yang

I. INTRODUCTION

Rabin fingerprints are short tags for large objects [1] that can be used in a wide range of applications, such as data deduplication, web querying, packet routing, and caching. We present a pipelined hardware architecture for computing Rabin fingerprint on high throughput streaming data. The design conducts real-time fingerprinting with short latencies, and can be tuned for optimized clock rate with a technique we call “split Fresh.” A pipelined sampling stage can select fingerprints based on pre-scheduling and only adds a few clock cycles of latency before returning the final result. The design can also be replicated to work in parallel for higher throughput data traffic. This architecture is implemented on a Xilinx Virtex-6 FPGA, and is put into test on a storage prototyping platform. The experiment results show that the design can achieve clock rates above 300 MHz and an order of magnitude improvement in latency over prior software implementations, while consuming little hardware resource. The scheme is extensible to other types of fingerprints and CRC computations, and is readily applicable to primary storage and caching in hybrid data storage systems.

II. ARCHITECTURE OUTLINE

Our design includes three main types of modules: Rabin fingerprint computation pipeline, channel sampling, and final selection, it is illustrated in Fig. 1, where the input data is from an 8-byte bus, and the shingles are also 8 bytes in size. of the channel units. After the sampling for a data chunk is done, the final selection unit will choose from the intermediate samples and returns a sketch for the data chunk [2].

A. Rabin Fingerprint Pipeline Design

Using a 64-bit wide data bus and a 64-bit shingle as an example, the pipeline design is illustrated in Fig. 1. The data is drawn from two consecutive clock cycles, where \( (a_0, a_1, \ldots, a_{63}) \) from the proceeding cycle, and \( (a_{64}, a_{65}, \ldots, a_{119}) \) from the following cycle.

Stage 0, the first stage in the pipeline, is a Fresh function, which computes its Rabin fingerprint out of \( (a_0, a_1, \ldots, a_{63}) \). The next seven stages are all Shift functions. The Shift function takes as an input the evicted byte from the previous shingle [3], the absorbed byte from the end of its own shingle, and the result from the previous shingle to produce a shingle of its own.

Fig. 3 shows a Rabin fingerprint pipeline design with the two split Fresh stages followed by 7 Shift stages. The two Fresh modules compute the fingerprint for the 8 bytes of data from the proceeding clock while Shift modules compute 7 shifts within every two consecutive 64-bit bus data shown in Fig. 2.

At steady state, an “FPn” is output at every stage to a channel sampling unit, and the entire module in our example produces eight fingerprints at every clock cycle.
B. Channel Sampling

Based on the sampling theory [4], each computed fingerprint is divided into two parts: index and signature, where the index is a few high order bits, and the signature the remaining ones. Four MSBs are the index, which is used to address the buffer where the selected signatures are stored. The comparator decides either the minimum or the maximum value is sampled into the buffer. The register is used to buffer the incoming signature to compare with the buffer output from the same high order bits’ signatures [5].

C. Final selection

When all signatures are settled in the buffer of the channel sampling module, the final selection unit can start to select the signatures to create the sketch. The behavior of the single pipeline design is shown using a shortened simulation session in Fig. 4. The “stage” signals display the fingerprint output from each stage, while the “signatures” signal is forming the final sketch. After the data transfer starts on the bus, it takes 8 clock cycles as the pipeline startup time for all the stages to work in parallel. At the end of the data transmission, the final selection takes action after another 8 clock cycles. Then, it takes 11 more clocks for all 8 signatures to be output. The total latency is therefore 19 clocks including the startup time and the final selection time, about 3.7% overhead for transmitting a 4KB data chunk.

In modern storage systems, fingerprinting is also implemented in software to take advantage of CPU power [8]. In order to evaluate the pros and the cons of software fingerprint computation and hardware solutions, we have implemented the software design presented in [7]. Our computation utilizes a sliding window based Rabin fingerprint library [8] to process the same data sets as the ones used in our hardware experiments.

Fig. 6 plots the latency as a function of the data block size when NVMe is 8 byte wide, 250MHz speed [9]. The software latency is measured from when the data chunk is ready in the memory to the finish of sketch generation. The hardware latency measurements begin with data arriving at the module and finishes after the result returns. The figure shows that our hardware implementation has a clear advantage with the latency difference increasing along the size of data blocks.

IV. REFERENCES