Multi-Level Hashing Dedup in HPC Storage Systems

Eric Valenzuela[1], Yin Lu[1] (Advisor), Susan Urban[2] (Advisor), Yong Chen[1] (Advisor)
[1] Department of Computer Science, [2] Department of Industrial Engineering, Texas Tech University
Texas Tech University 2014 NSF Research Experiences for Undergraduates Site Project

TEXAS TECH UNIVERSITY™

Abstract

- **Objectives**
  - To design a data deduplication system that provides 100% data integrity without a possibility of losing data, and
  - To introduce a multi-level data deduplication method to reduce costly byte-by-byte comparisons for 100% integrity
- **Background**
  - Reaching high ratios of data deduplication in High Performance Computing is highly achievable
  - Prior art demonstrates magnitudes of reduction possible and 15 to 30 percent of redundant data reduction on average
- **Challenges**
  - Existing deduplication systems can lose data due to hash collisions even though the probability is low
  - Byte-by-byte comparisons for all blocks can eliminate the potential data loss but are very costly
- **Contributions**
  - Designed a multi-level dedup method to reduce byte-by-byte comparisons while providing 100% data integrity
  - Implemented multi-level dedup while talking advantage of XeonPhi to compute cryptographic fingerprints concurrently
  - Proof-of-concept evaluations confirm promising results

Goal and Approach

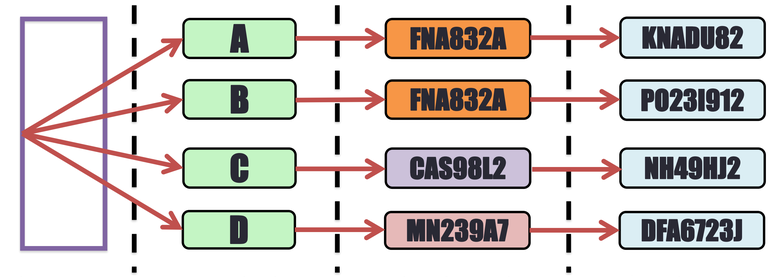
- **Xeon Phi**
  - Intel, Many Integrated Core Architecture (MIC)
  - Compute cryptographic fingerprints concurrently for multi-level data deduplication for rapid computations of hashes
- **OpenMP**
  - Xeon Phi uses OpenMP to compile parallel programs
  - With appropriate additions of **"pragma"** lines, the compiler interprets the program as parallel
- **Sequential Vs. Parallel**
  - Coded a sequential and parallel program in C that performs multi level hashing and dedup for large climate data sets
  - Compare the speed performance on Xeon Phi Nodes
- **Goal:**
  - Ensure no unique data is discarded in the process of data deduplication while maintaining exceptional performance

References

[1] Meister, D., Kaiser, J., Brinkmann, A., Cortes, T., Kuhn, M., & Kunkel, J. (2012, November). A study on data deduplication in HPC storage systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (p. 7). IEEE Computer Society Press.
[2] Broder, A., & Mitzenmacher, M. (2001). Using multiple hash functions to improve IP lookups. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*(Vol. 3, pp. 1454-1463). IEEE.
[3] Mark Ruijter. lessfs - open source data de-duplication. http://www.lessfs.com, February 2012. [Online; Consulted on June 23, 2014].
[4] Koutoupis, P. (2011). Data deduplication with Linux. *Linux Journal*, *2011*(207), 7.

Methodology

- **Multi-Level Hashing**
  - When two data blocks produce the same cryptographic fingerprint, we cannot immediately tell if both blocks are redundant or if a collision has occurred
  - Existing systems ignore collisions citing the probability is low and can lose data, intolerable especially for mission critical systems
  - We conduct byte-by-byte comparisons for 100% data integrity
  - We also introduce multi-level dedup to reduce costly comparisons and leverage parallel architectures to speed up hash calculations



- If several cryptographic fingerprints map to the same hash index, multiple hash functions are used as needed in data deduplication until we ensure the current data blocks being analyzed are in fact redundant

Although using low bit hash functions increases the chances for hash collisions, compared to secure hashing algorithms (2^{32} vs. 2^{512}), it has the potential to save memory when storing cryptographic fingerprints in the hash table

- **Proof-of-concept prototyping with Lessfs**
  - Open source inline data deduplication file system written for Linux
  - Lessfs, as most data deduplication file systems, do not implement byte-by-byte comparisons
  - Statistically, the hash functions provided are robust enough
  - Our motivation is to ensure no unique data is discarded as redundant data especially in reliable systems such as:
    - Aircrafts, space flight, military, and medicine

```
1720  /* Return the number of times this block is linked to files */
1721  unsigned long long getInUse(unsigned char *tigerstr)
1722  {
1723      unsigned long long counter;
1724      DAT *data;
1725
1726      loghash("getInuse search", tigerstr);
1727      if (NULL == tigerstr) {
1730      }
1731
1732      data = search_dbdata(DBU, tigerstr, config->hashlen, LOCK);
1733      if (NULL == data) {
1737      }
1738      memcpy(&counter, data->data, sizeof(counter));
1739      DATfree(data);
1740      LDEBUG("getInuse : return %llu", counter);
1741      return counter;
1742  }
1743
1744  void DATfree(DAT * data)
1745  {
1746      s_free(data->data);
1747      s_free(data);
1748      data = NULL;
1749  }
```

```
2237  unsigned int db_commit_block(unsigned char *dbdata,
2238                               INOBNO inobno, unsigned long dsize)
2239  {
2240      unsigned char *stiger = NULL;
2241      DAT *compressed;
2242      unsigned long long inuse;
2243      unsigned int ret = 0;
2244
2245      FUNC;
2246      LDEBUG("db_commit_block");
2247      stiger = thash(dbdata, dsize);
2248      create_hash_note(stiger);
2249      inuse = getInUse(stiger);
2250      if (0 == inuse) {
2256      } else {
2257          loghash("commit_block : only updated inuse for hash ", stiger);
2258      }
2259      inuse++;
2260      update_inuse(stiger, inuse);
2261      LDEBUG("db_commit_block : dbb %llu-%llu", inobno.inode,
2262             inobno.blocknr);
2263      bin_write_dbdata(DBB, (char *) &inobno, sizeof(INOBNO), stiger,
2264                       config->hashlen);
2265      delete_hash_note(stiger);
2266      s_free(stiger);
2267      return (ret);
2268  }
2269
```

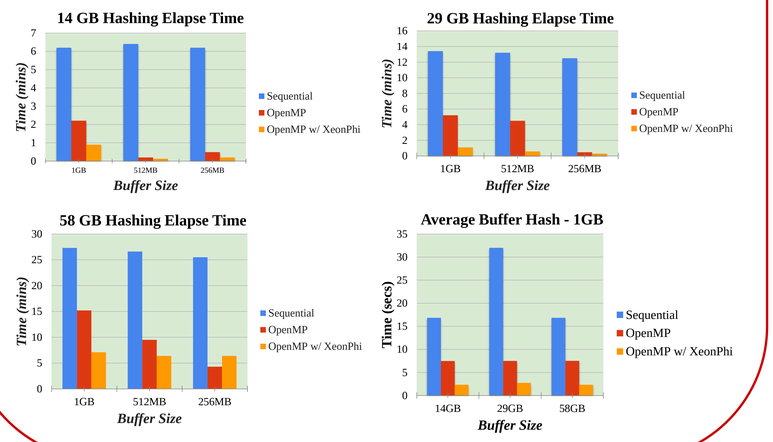
Two methods in Lessfs that are responsible for hashing the data blocks and determining whether the data blocks already exists

Performance Results

- **Tests with Climate Data Sets**
  - We performed four test runs by gathering three unique climate data sets from open source databases
  - Our results demonstrate that when using more threads in the parallel program, the elapsed time for hashing significantly drops
  - For instance, some sets results demonstrate:
    - Sequential = 27m34.689s *vs* Xeon Phi = 7m11.970s
  - However, different threads may be more appropriate at different times depending on the data size

The OpenMP code that was written can be compile and ran as a parallel program without taking advantage of the Xeon Phi's MIC's. However, an even greater performance was achieved with the combination of parallelism and Xeon Phi characteristics

- Three different buffer sizes were used: 1GB, 512MB, 256MB
- A 90% improvement was achieved while parallelizing the program
  - Hashing 1 GB buffer on average:16.83s - Sequential
  - Lowest average for hashing 1 GB buffer: 2.35s – OpenMP w/ Xeon Phi



Conclusion and Future Work

- Although our research performs an additional check for handling hash collisions while most data deduplication file systems do not, we highly emphasize the importance of not replying on probability and not losing data in data deduplication
- We provide this assurance by providing byte-by-byte comparisons and multi-level dedup technique
- We conduct proof-of-concept verification with open source data deduplication file system Lessfs and Xeon Phi

DISCLAIMER: This research is supported by the National Science Foundation and the Department of Defense under Grant No. CNS-1263183. We would like to thank the StackVelocity for their infrastructure support via the Cloud and Autonomic Computing site at Texas Tech University

STACK VELOCITY