5. OBSERVATIONS

AppDedupe is a prototype that can be implemented in user space using C++ and pthreads, on the Linux platform. The parallel deduplication efficiency is observed in the multicore deduplication server with real system implementation, while use trace-driven simulation to demonstrate how AppDedupe outperforms the state-of-the-art distributed deduplication techniques in terms of deduplication efficiency and system scalability. In addition, sensitivity studies can be conducted on chunking strategy, chunk size, super-chunk size, handprint size and cluster size.

5.1 Evaluation Platform and Workload:

Four commodity servers are used to perform the experiments to evaluate parallel deduplication efficiency in single-node dedupe server. All of them run Ubuntu 14.10 and use a configuration with 4-core 8-thread Intel X3440 CPU running at 2.53 GHz and 16GB RAM and a Seagate ST1000DM 1TB hard disk drive. In this prototype deduplication system, 7 desktops serve as the clients, one server serves as the director and the other three servers for dedupe storage nodes. It uses Huawei S5700 Gigabit Ethernet switch for internal communication. To achieve high throughput, our client component is based on an event driven, pipelined design, which utilizes an asynchronous RPC implementation via message passing over TCP streams. All RPC requests are batched in order to minimize the round trip overheads. We also perform event-driven simulation on one of the four servers to evaluate the distributed deduplication techniques in terms of deduplication ratio, load distribution, memory usage and communication overhead.

Five kinds of real world datasets are collected and two types of application traces for our experiments. The Linux dataset is a collection of Linux kernel source code from versions 1.0 through 3.3.6, which is downloaded from the website. The VM dataset consists of 2 consecutive monthly full backups of 8 virtual machine servers (3 for Windows and 5 for Linux). The audio, video and photo datasets are collected from personal desktops or laptops. The mail and web datasets are two traces collected from the web server and mail server of the CS department in FIU. The key workload characteristics of these datasets are summarized in Table 5.1. Here, the "size" column represents the original dataset capacity, and "deduplication ratio" column indicates the ratio of logical to physical size after deduplication with 4KB fixed chunk size in static chunking (SC) or average 4KB variable chunk size in optimized content defined chunking (CDC) based on the open source code in Cumulus.

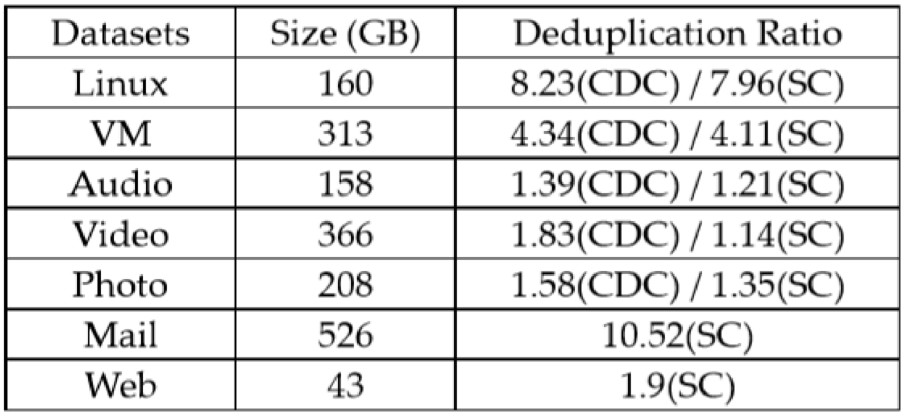
5.2 Evaluation Metrics:

The following evaluation metrics are used in our evaluation to comprehensively assess the performance of our prototype implementation of AppDedupe against the state-of-the-art distributed deduplication schemes.

Deduplication efficiency (DE): It is first defined to measure the efficiency of different dedupe schemes in the same platform by feeding a given dataset.

TABLE 5.1

The Workload Characteristics of the Real-World Datasets and Traces



Datasets	Size (GB)	Deduplication Ratio
Linux	160	8.23(CDC) / 7.96(SC)
VM	313	4.34(CDC) / 4.11(SC)
Audio	158	1.39(CDC) / 1.21(SC)
Video	366	1.83(CDC) / 1.14(SC)
Photo	208	1.58(CDC) / 1.35(SC)
Mail	526	10.52(SC)
Web	43	1.9(SC)

It is calculated by the difference between the logical size L and the physical size P of the dataset divided by the deduplication process time T. So, deduplication efficiency can be expressed through the following expression (7).

$$DE = \frac{L-P}{T} \qquad (7)$$

Normalized deduplication ratio (NDR): It is equal to the distributed deduplication ratio (DDR) divided by the single-node deduplication ratio (SDR) achieved by a single-node, exact deduplication system, and can be expressed in (8). This is an indication of how close the deduplication ratio achieved by a distributed deduplication method is to the ideal distributed deduplication ratio.

$$NDR = \frac{DDR}{SDR} \qquad (8)$$

Normalized effective deduplication ratio (NEDR): It is equivalent to normalized deduplication ratio divided by the value of 1 plus the ratio of standard deviation σ of physical storage usage to average usage α in all dedupe servers. Normalized effective deduplication ratio can be expressed in (9). It indicates how effective the data routing schemes are in eliminating the deduplication node information island.

$$NEDR = \frac{DDR}{SDR} \times \frac{\alpha}{\alpha+\sigma} \qquad (9)$$

Number of fingerprint index lookup messages: It includes that of inter-node messages and intra-node messages for chunk fingerprint lookup, both of which can be easily obtained in our simulation to estimate communication overhead.

RAM usage for intra-node deduplication: It is an essential system overhead related to chunk index lookup in dedupe server. And it indicates how efficient the chunk index lookup optimization is to improve the performance of intra-node deduplication.

Data skew for distributed storage: DS is used as a metric for data skew in the dedupe storage server cluster. It can be expressed in (10), and equals to the difference between the maximum capacity *MaxLoad* and the minimum capacity *MinLoad* in storage cluster divided by the mean value *MeanLoad*.

$$DS = \frac{MaxLoad - MinLoad}{MeanLoad} \qquad (10)$$

5.3 Application-aware Deduplication Efficiency:

Here the client performs data partitioning and chunk fingerprinting in parallel before data routing decision. It can divide the files into small chunks with fix-sized SC or variable-sized CDC chunking methods for each kind of application files, and calculates the chunk fingerprints with cryptographic hash function. Then, hundreds of or thousands of consecutive smaller chunks are grouped together as a super-chunk for data routing. The implementation of the hash fingerprinting is based the OpenSSL library. According to the study in SHA-1 is reduce the probability of hash collision for fix sized SC chunking, while we choose MD5 for variable-sized CDC chunking for high hashing throughput with almost the same hash collision possibility. To exploit the multi-core or many-core resource of the dedupe storage node, we also develop parallel application-aware similarity index lookup in individual dedupe servers. Here multiple data stream based parallel deduplication, each data stream has a deduplication thread, but all data streams share a common hash-table based application aware similarity index in each dedupe server. The hash table is locked based application-aware similarity index by partitioning the index at the application granularity to support concurrent lookup. The single-node parallel deduplication perform the best in application-aware-similarity-index lookup when the number of data streams equals to that of supported CPU logical cores, while the performance of more streams drops when the number of locks is larger than the number of data stream because of the overhead of thread context switching that causes data swapping between cache and memory. In this application aware similarity index with the traditional similarity index for parallel deduplication throughput in single dedupe storage node with multiple data streams. The results in Fig. 5.1 show the parallel deduplication throughput using the VM dataset, with data input from RAMFS to eliminate the performance interference of the disk I/O bottleneck. The throughput of both traditional similarity index (Naive) and application aware similarity index (Application aware) with cold cache or warm cache respectively will be tested to get the results.

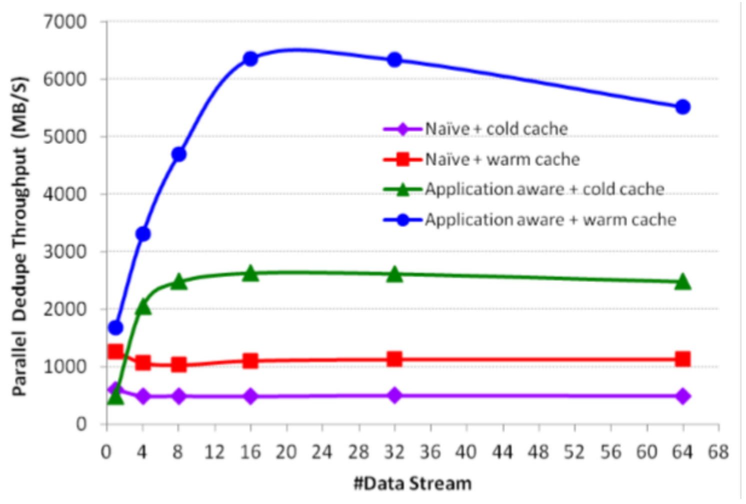


Fig.5.1 The Comparison of parallel deduplication throughput of the application aware similarity index structure with that of the traditional similarity index structure for multiple data streams.

Here, "cold cache" means the chunk fingerprint cache is empty when we first perform parallel deduplication with multiple streams on the VM dataset. While "warm cache" means the duplicate chunk fingerprint had already been stored in the cache, when we perform parallel deduplication with multiple streams again on the same dataset. We observe that the parallel deduplication schemes with application-aware similarity index perform much better than the naïve parallel deduplication mechanisms, and the parallel deduplication schemes with warm cache can achieve higher throughput than those regimes with cold cache. The throughput of the parallel deduplication with both application aware similarity index and warm cache goes up to 6.2GB/s with the increasing number of data streams; After 16 concurrent streams, the throughput falls to 5.5GB/s since the concurrency overheads of index locking and disk I/O are becoming obvious. The deduplication efficiency is measured in a configuration with two clients, one director and a single dedupe storage node to show the tradeoff between deduplication effectiveness and system overhead. To eliminate the impact of the disk bottleneck, we store the entire workload in memory and perform the deduplication process to skip the unique-data-chunk store step. To assess the impact of the system overhead on deduplication efficiency, we measure "Bytes Saved Per Second", as a function of the chunk size.

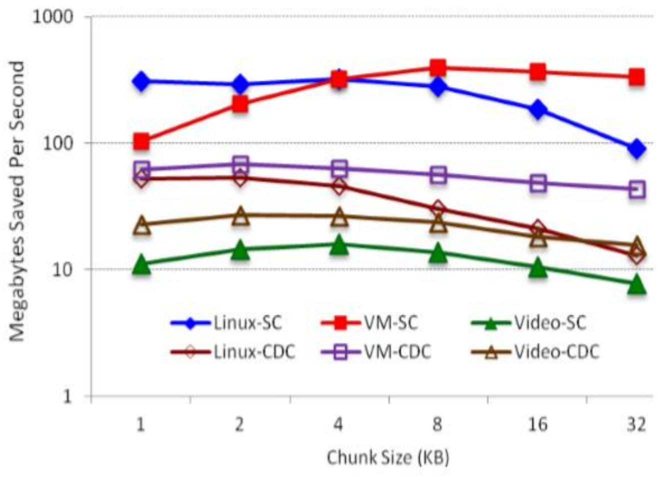
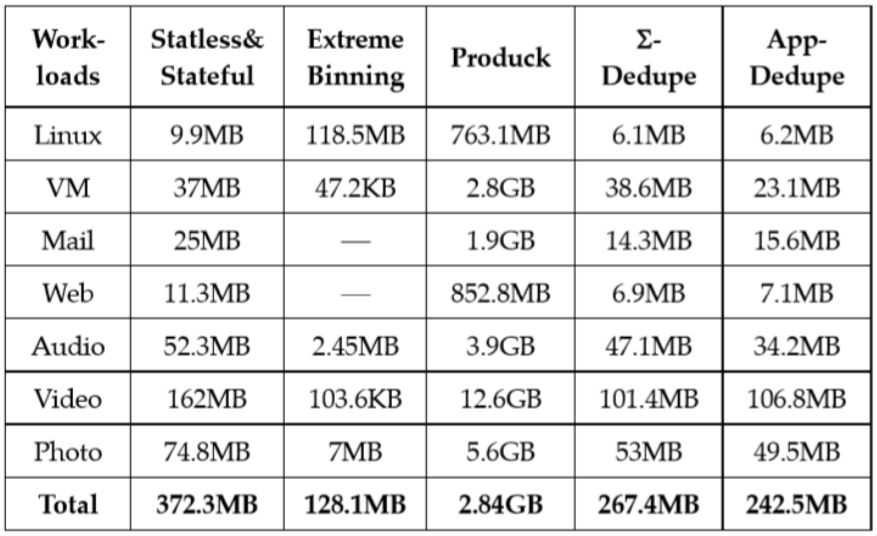


Fig.5.2 Deduplication efficiency in single storage node.

The results in Fig. 5.2 show that the best choices of chunking method and chunk size to achieve high deduplication efficiency vary with different application datasets. The single dedupe server can achieve the highest deduplication efficiency when the chunk size is 4KB for statically chunked Linux workload, 8KB for statically chunked VM workload and 2KB for Video workload with CDC. As a result, we choose to perform application-aware chunking, which adaptively select the best chunking scheme and the best chunk size for each application with high deduplication efficiency. Since the disk based chunk fingerprint index is always too large to fit it into the limited RAM space, an efficient RAM data structure is needed to improve the index lookup performance and keep high deduplication accuracy. In EMC super-chunk based routing schemes, Bloom Filter is employed to improve its full index lookup performance. A file-level in-RAM primary index over chunk-level on-disk indices is provided in Extreme Binning to achieve excellent RAM economy. In AppDedupe, an application-aware similarity index is designed in RAM to speed up the chunk fingerprint index lookup process. To compare the RAM overhead of these three intra-node deduplication methods, we show their RAM usage on the seven different application workloads in Table 5.2, to eliminate more than 95% duplicate chunks, with 8KB mean chunk size in the Bloom Filter design, 48B primary index entry size for Extreme Binning, and 40B entry size for chunk index of Produck and application-aware similarity index of our design.

TABLE 5.2

Ram usage of Intra-Node Deduplication for various application workloads



Work-loads	Statless& Stateful	Extreme Binning	Produck	Σ-Dedupe	App-Dedupe
Linux	9.9MB	118.5MB	763.1MB	6.1MB	6.2MB
VM	37MB	47.2KB	2.8GB	38.6MB	23.1MB
Mail	25MB	—	1.9GB	14.3MB	15.6MB
Web	11.3MB	—	852.8MB	6.9MB	7.1MB
Audio	52.3MB	2.45MB	3.9GB	47.1MB	34.2MB
Video	162MB	103.6KB	12.6GB	101.4MB	106.8MB
Photo	74.8MB	7MB	5.6GB	53MB	49.5MB
Total	**372.3MB**	**128.1MB**	**2.84GB**	**267.4MB**	**242.5MB**

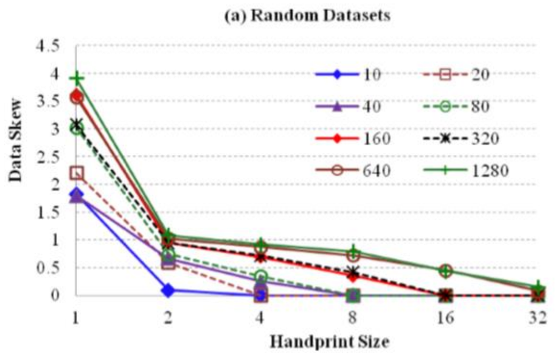
The RAM usage cannot be estimated for Extreme Binning on the two traces (i.e., Mail and Web) without file metadata information. It is clearly that our design outperforms the traditional index lookup design of Produck, and the Bloom Filter design of EMC stateless and stateful schemes in shrinking the RAM usage for our sampling based index design. AppDedupe also outperforms the previous $\sum -$Dedupe design with less RAM usage due to its dynamic chunk size selection in application-aware chunking scheme. Our application-aware similarity index occupies more space than file-level primary index in Extreme Binning for datasets with large files, like VM and multimedia workloads, but less for Linux source code dataset with small files. We can further reduce its size by adjusting super-chunk size or handprint size with the corresponding deduplication effectiveness loss.

4.4 Load Balance in Super-Chunk Assignment:

Load balance is an important issue in distributed storage technique. It can help improve system scalability by ensuring that client I/O requests are distributed equitably across a group of storage servers. The implementation of consistent hashing based DHTs in traditional distributed deduplication are considerable load imbalance due to its stateless assignment design.

In particular, a storage node that happens to be responsible for a larger segment of the keyspace will tend to be assigned a greater number of data objects. DS is a metric for data skew in the storage server cluster. Data assignment method can achieve global load balance when DS=0. The larger DS value we measured in the test, the more serious load imbalance happened in the storage cluster. To make our conclusion more general, we not only consider an ideal scenario that each super-chunk is filled with random data in Fig. 5.3(a), but also perform the tests on real-world datasets: VM images, Linux source code, and mail datasets with 10 and 1280 nodes, respectively, as shown in Fig. 5.3(b). Super-chunk chooses the least loaded node in the k storage nodes that are mapped independently and uniformly at random by consistent hashing with the k representative fingerprints in its handprint.

The effectiveness of load balancing has been tested with different handprint size and node number in Fig. 5.3. The average load of each node is 65536 super-chunks with 1 MB size. The important implication of the results is that even a small amount of representative fingerprints in handprint can lead to drastically different results in load balancing. It shows that traditional schemes, like DHT or hash-bucket, with only one representative fingerprint for choice are hard to keep a good load balance in the super-chunk assignment. When the cluster scales to hundreds or thousands of nodes, our handprint technique can keep a good load balance (the metric of data skew is less than 1) by routing the super-chunks with $k \geq 4$ random choices. We select handprint size from 4 to 16 to make a tradeoff between load balance and communication overhead for large-scale distributed deduplication in big data.



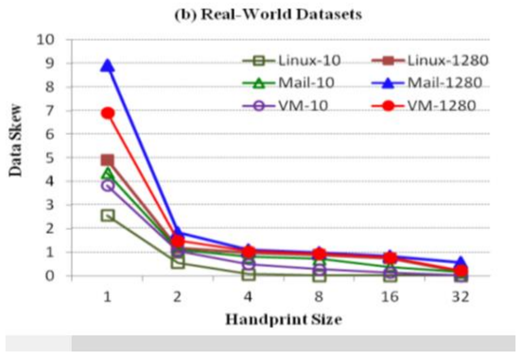


Fig.5.3 The data skew of super-chunk routing when we select different handprint sizes under various node numbers in the storage server cluster.

4.5 Distributed Deduplication Efficiency:

The data is routed at the super-chunk granularity to preserve data locality for high performance of distributed deduplication, while performing deduplication at the chunk granularity to achieve high deduplication ratio in each server locally. Since the size of the super-chunk is very sensitive to the tradeoff between the index lookup performance and the distributed deduplication effectiveness, the super-chunk size of 1MB is considered to reasonably balance the conflicting objectives of cluster-wide system performance and capacity saving, and to fairly compare our design with the previous EMC distributed deduplication mechanism.

In this section, a sensitivity study is conducted to select an appropriate handprint size for our AppDedupe scheme, and then compare our scheme with the state of the art approaches that are most relevant to AppDedupe, including EMC's super-chunk based Stateful and Stateless data routing, ∑-Dedupe, Produck and Extreme Binning, in terms of the effective deduplication ratio, normalized to that of the traditional single-node exact deduplication, and communication overhead measured in number of fingerprint index lookup messages. Each node is emulated by a series of independent fingerprint lookup data structures, and all results are generated by trace-driven simulations on the seven datasets under study. Two-tiered data routing can accurately direct similar data to the same dedupe server by exploiting application awareness and data similarity. A series of experiments are done to demonstrate the effectiveness of distributed deduplication by our handprint-based deduplication technique with the super-chunk size of 1MB on the Linux kernel source code workload. Fig. 5.4 shows the deduplication ratio, normalized to that of the single-node exact deduplication, as a function of the handprint size.

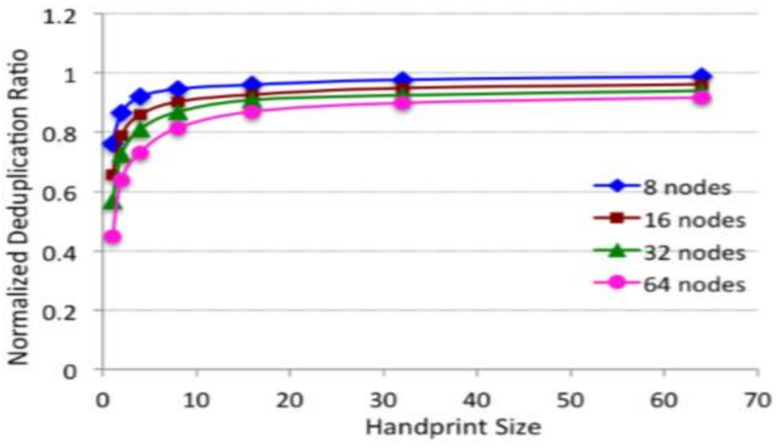


Fig.5.4 Distributed deduplication ratio normalized to that of singlenode exact deduplication with various cluster size, as a function of handprint size

As a result, AppDedupe becomes an approximate deduplication scheme whose deduplication effectiveness nevertheless improves with the handprint size because of the increased ability to detect similarity in super chunks with a larger handprint size. It is observed that there is a significant improvement in normalized deduplication ratio for all cluster sizes when handprint size is larger than 8. This means that, for a large percentage of super-chunk queries, we are able to find the super-chunk that has the largest content overlap with the given superchunk to be routed by our handprint-based routing scheme. To strike a sensible balance between the distributed deduplication ratio and system overhead, and match the handprint size choice in single-node, a handprint consisting of 8 representative fingerprints is considered in the following experiments to direct data routing on super-chunks of 1MB in size. After the selection on super-chunk size and handprint size, we conduct prelimary experiments on AppDedupe prototype in small scale to compare it with the two distributed deduplication schemes with application-awarerouting and hand-print-based-statefulrouting (∑-Dedupe) respectively.

The distributed dedupe storage system is fed with the chunk fingerprints of the above described seven application workloads totalled 1774 GB size, to eliminate the disk I/O bottleneck. In Table 5.3, application distribution is also defined as the number of application type distributed in three dedupe storage nodes, record time spent for the deduplication processes and indicate capacity to describe the total storage capacity after distributed deduplication in the storage cluster. The results in key metrics are shown that our AppDedupe performs best in deduplication efficiency with high deduplication ratio and low time overhead.

Table.5.3 Key Metrics of the Distributed Deduplication Schemes

Method	Application Distribution	Capacity (GB)	DS	EDR	Time(s)	DE (MB/s)
AppAware	2:2:3	610	1.84	0.45	28335	41.8
Σ-Dedupe	7:7:7	657	0.05	0.87	30250	36.6
AppDedupe	3:4:4	618	0.07	0.95	23775	48.6

AppAware scheme has the highest deduplication ratio, but it suffers from load imbalance due to the data skew of application level data assignment. ∑-Dedupe suffers from a low deduplication ratio since it distributes all types of application data into each dedupe storage node with the highest cross-node redundancy. To show the scalability of AppDedupe design, simulations are performed to compare it with the existing data routing schemes in distributed deduplication in large scale. AppDedupe scheme is compared with the state-of-the-art data routing schemes of Extreme Binning, Produck, ∑-Dedupe, EMC's stateless routing scheme and stateful routing scheme, across a range of datasets. Fig. 5.5 plots EDR as a function of the cluster size for the six algorithms on all datasets. Because the last two traces, Mail and Web, do not contain file-level information, we are not able to perform the file-level based Extreme Binning scheme on them.

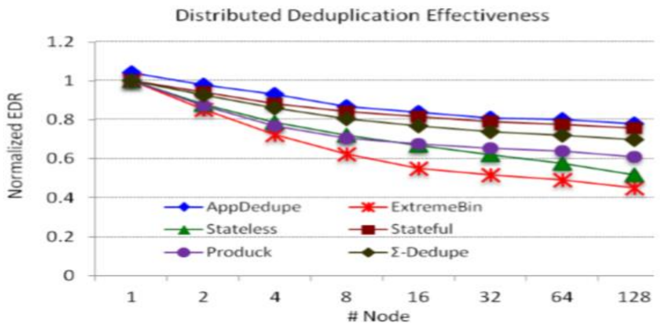


Fig.5.5 Effective deduplication ratio (EDR), normalized to that of single-node exact deduplication, as a function of cluster size on workloads.

App-Dedupe can achieve the highest effective deduplication ratio due to its load balancing design of inter-node two-tiered data routing and high deduplication effectiveness of intra-node application-aware chunking by leveraging application awareness and data similarity. More specifically, the AppDedupe scheme achieves 103.1% of the EDR obtained by the Stateful scheme for a cluster of 128 server nodes on the seven datasets, while this performance margin narrows to 103.6% when averaging over all cluster sizes, from 1 through 128. Stateless routing performs worse than AppDedupe, Produck and Stateful routing due to its low cluster-wide data reduction ratio and unbalanced capacity distribution. Extreme Binning underperforms Stateless routing on the workloads because of the large file size and skewed file size distribution in the datasets, workload properties that tend to render Extreme Binning's similarity detection ineffective. Produck achieves higher normalized EDR than Stateless routing and Extreme Binning due to its stateful routing design, but it underperforms AppDedupe on all datasets for its low deduplication ratio and unbalanced load distribution.

AppDedupe outperforms Extreme Binning in EDR by up to 72.7% for a cluster of 128 nodes on the five datasets containing file level information. For the seven datasets, AppDedupe is better than Stateless routing in EDR by up to 50.5% for a cluster of 128 nodes. AppDedupe achieves an improvement of 28.2% and 11.8% in EDR with respect to Produck and ∑-Dedupe on all datasets in a cluster of 128 nodes, respectively. As can be seen from the trend of curves, these improvements will likely be more pronounced with cluster sizes larger than 128.

In distributed deduplication storage systems, fingerprint lookup tends to be a persistent bottleneck in each dedupe storage server because of the costly on-disk lookup I/Os, which often adversely impacts the system scalability due to the consequent high communication overhead from fingerprint lookup. To quantify this system overhead, we adopt the number of finger print lookup messages as a metric. This metric is measured by totaling the number of chunk fingerprint-lookup messages on the seven datasets, for the five distributed deduplication schemes. As shown in Fig. 5.6 that plots the total number of fingerprint-lookup messages as a function of the node number, AppDedupe, Extreme Binning and Stateless routing have very low system overhead due to their constant fingerprint-lookup message count in the distributed deduplication process, while the number of fingerprint-lookup messages of Stateful routing grows linearly with the node number.

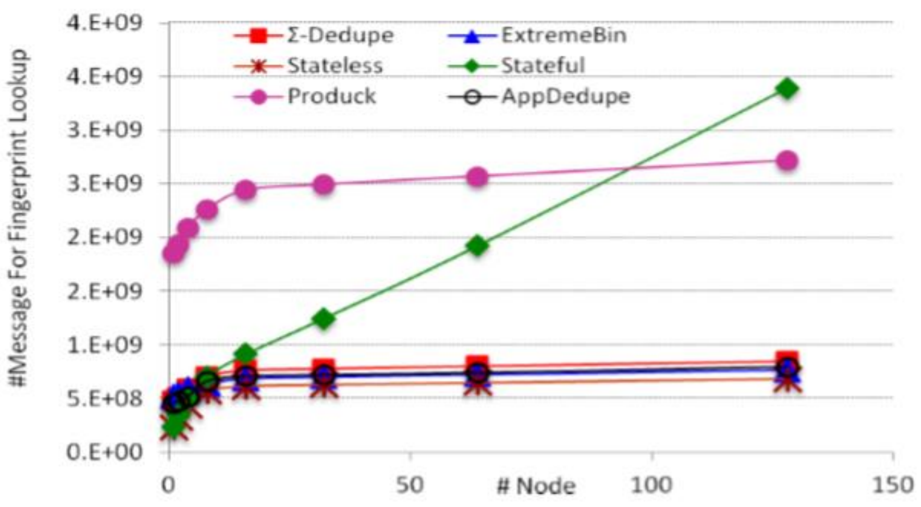


Fig.5.6 Communication overhead in terms of the number of fingerprint-lookup message for all seven workloads.

This is because Extreme Binning and Stateless routing only have 1-to-1 client-and- server fingerprint lookup communications for source deduplication due to their stateless designs. Stateful routing, on the other hand, must send the fingerprint lookup requests to all nodes, resulting in 1-to-all communication that causes the system overhead to grow linearly with the node number even though it can reduce the overhead in each node by using a sampling scheme. Produck has high communication overhead due to its fine-grained chunk size with 1KB, while other deduplication methods adapt 4KB or 8KB.

The number of fingerprint-lookup messages in Produck is about four times that of AppDedupe, Extreme Binning and Stateless routing, and it grows as slow as these three low-overhead schemes. As described in Algorithm 1, the main reason for the low system overhead in AppDedupe is that the pre-routing fingerprint lookup requests for each super-chunk only need to be sent to at most 8 candidate nodes, and only for the lookup of representative fingerprints, which is 1/32 of the number of chunk fingerprints, in these candidate nodes. The message overhead of AppDedupe in fingerprint lookup is about 1.25 times that of Stateless routing and Extreme Binning in all scales. ∑-Dedupe is the preliminary version of our AppDedupe, and they have almost the same communication overhead due to their consistent interconnect protocol.