BORA: A Bag Optimizer for Robotic Analysis

Jian Zhang*§¶‖, Tao Xie†, Yuzhuo Jing*, Yanjie Song*, Guanzhou Hu*, Si Chen‡, and Shu Yin*§¶‖x
*School of Information Science and Technology, ShanghaiTech University, Shanghai, China
Email: {zhangjian, jingyzh, songyj, hugzh1, yinshu} @shanghaitech.edu.cn
†Department of Computer Science, San Diego State University, CA 92182, USA Email: txie@sdsu.edu
‡Department of Computer Science, West Chester University, PA 19383, USA Email: schen@wcupa.edu
§Shanghai Engineering Research Center of Intelligent Vision and Imaging, China
¶Chinese Academy of Sciences, Shanghai Institute of Microsystem & Information Technology, China
‖University of Chinese Academy of Science, China
xCorresponding Author

***Abstract*—We present BORA (Bag Optimizer for Robotic Analysis), a file system middleware that optimizes the acquisition of bags, which are specially formatted files used to store timestamped ROS (robot operating system) messages. BORA sits between ROS and an existing file system to conduct semantic-aware data pre-processing. In particular, it categorizes ROS bag data into multiple groups with each having a distinct label. BORA predigests data index constructions and reduces file open time via a hash-based label management scheme. It is also capable of providing ROS analytic applications with only data needed without a sequence of data searching and locating operations. We implement a BORA prototype, which is then integrated into three computing platforms: a single-node server, a four-node PVFS storage cluster, and a Tianhe-1A Supercomputer storage subsystem. Next, we evaluate the BORA prototype on the three platforms using four real-world ROS applications. Our experimental results show that compared to a traditional bag management scheme BORA improves data acquisition performance by up to 11x. In addition, it offers up to 10x data acquisition performance improvement and 3,100x bags open improvement under a swarm robotics data analysis scenario where data is retrieved across multiple bags simultaneously.**

I. Introduction

Robot operating system (ROS) is an open-sourced meta-operating system framework that consists of libraries and tools to help software developers create robotic applications such as simultaneous localization and mapping (SLAM) [1], grasping [2] and navigation [3].

ROS works alongside a traditional operating system like Linux to facilitate developers to resolve some specific issues (e.g., distributed computation, software reuse, and rapid testing) that appear in the development of robotic software. Besides conventional robotic applications, robotic control and analysis systems such as industrial robotics [4], UAV (unmanned aerial vehicle) swarms [5], and low-power rescue devices [6] are also developed on top of ROS.

The abstractions provided by ROS allow developers to design and implement robotic applications without considering underlying systems. A robot control system usually comprises multiple processes that perform computation. Each process is called a *node* in ROS. For example, one node controls a laser range-finder while another node controls wheel motors. Nodes communicate with each other by passing *messages*, which are routed via logical publish/subscribe buses called *topics* [7]. A node sends out a message by *publishing* it to a given topic, whereas a node that is interested in a certain kind of data will *subscribe* to an appropriate topic [7]. In general, publishers and subscribers are not aware of each others' existence, which decouples information producers from consumers. Fig. 1a shows the ROS stack. ROS provides a tool called `rosbag` that can record the messages published on one or more topics to a specially formatted file called *bag* and then replay those messages later. The message recording and replaying capabilities form a powerful way to test some robot software: a developer can run a robot only a few times while recording some relevant topics, and then replay the messages on those topics many times to experiment with the software that processes those data [7]. Serving as a fundamental storage abstraction in the ROS framework, bag works well as it reaches its original expectation.

However, more than just replaying messages many applications nowadays need to extract messages of certain topics from bag files for later analysis [8] [9]. For example, SLAM needs to extract image data from bag files to build a point cloud and further generate a map based on inertial measurement data [1]. To achieve this goal, the `rosbag` tool needs to perform a sequence of operations including scanning the offset of *chunk* sections, collecting data types and message definitions, as well as locating and retrieving message records via `seek` operations. Besides, in order to provide two-dimensional data queries such as (topics, time_range), `rosbag` has to collect all the timestamps of message records and then build a tree-structure to target the requested messages in a range between start_time and stop_time [7]. The current way of extracting messages from bag files is inefficient due to the following reasons: (1) each time a developer opens a bag file, ROS needs to first scan it to gather message location information and statistics for high-level indexing, which is a time-consuming repeated effort; (2) a developer must write a script to either replaying a bag file to subscribe the messages of interests or iterate over messages in the bag, which is neither efficient nor reusable; (3) since bag is not originally designed to handle complex queries for data analysis, it is very time-consuming to extract message records of multiple topics, especially when

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

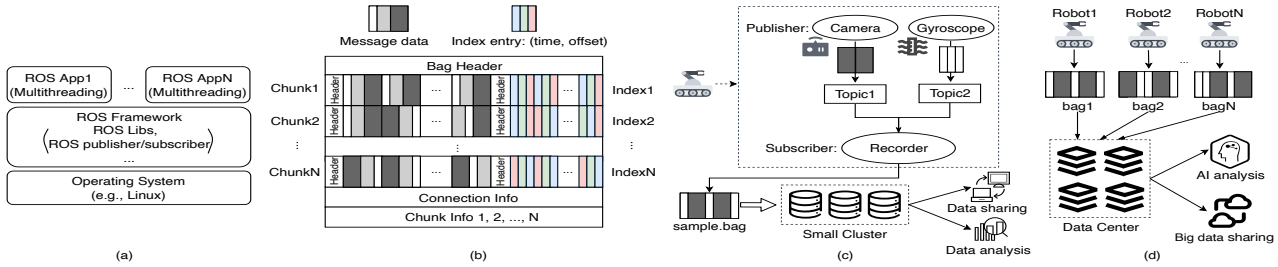

Fig. 1: (a) ROS stack; (b) bag format; (c) an example of a single-robot system; (d) an example of a swarm of robots system.

the size of a bag file keeps growing.

To provide rich query capabilities, some studies suggest that a database system could be used to replace the current bag file mechanism [10]. We argue, however, the advantages of a file system can better fit the needs of ROS data storage and management. First of all, a ROS bag file has a native ability to quickly store a large volume of data in a chronological order, which is essential for robotic applications [11]. Second, the ROS bag file abstraction ensures prompt data migration either from robots to a desktop computer or from local computers to a remote cluster [12]. Third, a ROS bag file can store poly-type data including structured data (e.g., GPS locations, inertial measurements, pressures, etc.) and unstructured data (e.g., images, laser scans, videos, etc.) [13]. On the contrary, it is hard for a database system to accomplish this task. Finally, a file system ensures that ROS can concurrently collect multiple bag files from swarm robots and provide parallel cross-bag data retrieving capabilities.

To solve the problem of inefficient ROS message acquisition while retaining the advantages of the bag mechanism, in this paper we present BORA (Bag Optimizer for Robotic Analysis). BORA is essentially a file system middleware that optimizes ROS bag data storage and acquisition. It sits between the ROS framework and an underlying file system to conduct topic-conscious bag data re-organization on a storage node. Based on the components of a bag, BORA is able to re-organize its data into multiple groups with each having a distinct label and manage these data groups through a hash-based index. Thus, BORA can provide ROS applications with a well-organized data layout and a simpler bag data locating scheme. Besides, it provides a coarse-grain secondary index mechanism to provide data that meets some particular period predicates. As a result, BORA can reduce bag open time and increase the efficiency of interested data retrieving.

The main contributions of this paper include: (1) we propose BORA, a first file system middleware to optimize data organization of ROS bags for robotic analysis. BORA is designed to enhance data query capability of existing bag-based ROS storage systems with a minimum cost; (2) we implement a BORA prototype and then integrate it into three computing platforms (i.e., a single-node server, a four-node cluster, and a Tianhe-1A storage subsystem under four real-world robotic applications; (3) a comprehensive experimental study is provided to fully evaluate the efficacy of the BORA prototype; (4) BORA can be readily extended to most robotic data analytic and implementation applications, which also require a similar data extracting and locating procedure. We plan to release the source code of BORA for public use.

The rest of the paper is organized as follows. Section II provides the background and motivation of this research. The design and implementation details of BORA are presented in Section III, which is followed by an evaluation of BORA shown in Section IV. While Section V discusses some lessons learned from this research, Section VI summarizes the related work. Finally, Section VII concludes this paper.

II. Background and Motivation

In this section, we first briefly introduce bag, which is the storage mechanism of ROS. Next, we explain the differences in bag data processing between the existing way and our proposed BORA-assisted approach.

A. Overview of ROS Bag

A bag is a file format in ROS for storing message data. A message is a simple data structure that comprises multiple typed fields. All bags are files with a .bag extension. An array of tools have been built to allow a developer to store, process, analyze, and visualize them [7]. Bags are typically created by a tool like `rosbag`, which subscribes to one or more ROS topics and stores serialized message data in a file as it is received. Fig. 1b shows a typical ROS bag file format. A bag is comprised of an array of diverse records including bag header, chunk, index, connection, and chunk info. The bag header stores information about the entire bag (e.g., the offset to the first index record and the number of chunks) [7]. The `rosbag` tool stores messages in the unit of chunk and appends an index record to each chunk. An index record contains offsets and timestamps of all messages in the preceding chunk as shown in Fig. 1b. Index data is scattered all over a bag. The connection info record contains metadata about a connection being established, including typing information and routing information. After all messages are recorded, `rosbag` generates a chunk info record for each chunk and then appends them to the end of the bag. A chunk info record stores the offset of its corresponding chunk and timestamps of the earliest and the latest messages in that chunk. A detailed description of bag format can be found in [7].

There are two typical ways of using bags: (1) online use in a ROS computation graph where a developer can collect

certain message data in a bag on the fly to control real-time robots; (2) offline use in data replaying where a developer can echo data on screens in the time order of message collecting. A ROS computation graph is a peer-to-peer network of ROS processes that are processing data together [7]. Fig. 1c shows a simple ROS computation graph. There are some APIs like `rebagging` available for developers to iterate over a bag and extract messages that match a particular filter into a new bag file. Furthermore, some tools allow developers to automatically update bag files when messages are out of date [7].

Fig. 1c also illustrates an example of a single-robot ROS system. In this system, the Camera node and Gyroscope node publish messages to two different topics (i.e., Topic1 and Topic2), respectively. After the command `rosbag record -O sample.bag Topic1 Topic2` is executed, `rosbag` creates a new node called Recorder, which subscribes to the two topics (see Fig. 1c). It then writes all messages received into a file called sample.bag. The amount of data collected in a bag file varies from a few hundred megabytes to hundreds of gigabytes or even larger. The sample.bag file is typically stored in a workstation or a small cluster. It can be then analyzed in-situ or shared with a remote server. In addition to many single-robot systems, some ROS applications demand an autonomous robot swarm, a bio-inspired concept that provides a robust and flexible robotic system by exploiting a large number of robots. This concept allows for the coordination of robots in order to cooperatively perform a single global task. Fig. 1d shows an example of a robotic swarm. Each of them generates a bag file, which is then dumped to a data center where a sufficient storage capacity and computational power are available to store and analyze the huge amount of data collected.

B. New Requirements of ROS Bag Usage

With rapid development of artificial intelligence in recent years, ROS applications with advanced data analysis requirements are emerging [14] [15] [16]. For example, SLAM needs to extract vision data from bags to build point clouds, and then, generate an environmental map with landmark data from bags to help robots locate themselves in an unknown territory [9]. Object and pattern recognition techniques require to extract RGB image data, image data with depth info, and camera pose info from bags to perform training of object models, which are then used for detection [17]. Assisted by a spectrum of new artificial intelligence techniques, ROS developers are applying various deep learning algorithms such as CNN (convolutional neural networks), RNN (recurrent neural networks), and GAN (generative adversarial networks), to train models for pattern recognition [18], decision making [19], and controls on robots [20]. These algorithms require a set of environmental information as inputs, which leads to the need of retrieving a significant amount of structured and unstructured data from bags. How to efficiently extract data from bags for model training becomes a new challenge. The main reason is that bags were originally designed in a log-structured way to quickly record sensor data in chronological order. The *rosbag* tool needs to scan the entire bag to locate the scattered messages and iterate over them to find the wanted data. Besides, the performance potential of underlying storage devices cannot be fully exploited as a file system rarely has semantic information from applications to optimize its I/O.

To meet the data analysis requirements of these emerging ROS applications, some researchers proposed to replace existing bag mechanism with a sophisticated database management system (DBMS) in order to enhance the performance of queries for ROS data analysis [21]. We argue that a database-backed message logging system might be inadequate to ROS applications. First, the current ROS publisher/subscriber mechanism supports a data rate of GB/s when running an advanced robot such as PR2 [22]. A DBMS, however, can hardly collect continuous large volumes of data in real-time. Second, robot developers often store robotic data on various computing platforms with distinct storage capabilities ranging from embedded devices, laptops, to clusters [10]. A DBMS may be unsuitable for an embedded computing platform. Third, robotic data consists of both structured data (joint angles, transpose vectors, altitude, latitude, etc.) and unstructured data (laser scans, images, motion pictures, etc.) [11]. However, a DBMS might not be able to support a rich querying interface to deal with mixed data structures.

To verify whether a DBMS is an appropriate substitute for an existing file system utilizing the ROS bag mechanism, we conducted a preliminary experiment on an Intel workstation with 16 GB main memory and two 256 GB NVMe SSDs. In particular, we measured the performance of three representative database systems (i.e., a NoSQL database called Areospike [23], a traditional SQL DB named PostgreSQL [24], and a time series database called InfluxDB [25]) and one file system (i.e., Ext4) when 49,233 TF (transform stamped info) messages were inserted (see Fig. 2). The TF messages were extracted from a real-world Handheld SLAM bag [26]. We found that it only took Ext4 130 milliseconds to append 49,233 TF messages to a bag file. To insert the same set of TF messages, the three database systems are 51.8x, 93.6x, and 3,694.6x slower than Ext4, respectively. In addition, InfluxDB cannot support complex array structures, and thus, is inadequate to process ROS data, which could be multiple dimensional. For example, an IMU (inertial measurement unit) message contains four float64 data structures with each consisting of a three-dimensional array [26]. A DBMS also has some limitations in sharing data among systems with diverse configurations. The poor message insertion performance of the database systems plus their inadequacies in ROS data processing motivated us to stick to the path of development of a file system middleware to enhance the performance of data acquisition of bags.

Motivation 1: there is a great need to develop a file system middleware that can optimize ROS bag organization so that the overhead of data traversal can be largely reduced while the original useful bag abstraction can be retained. In this way, robots can store and share a high-volume of data quickly after the data has been collected. Existing I/O middleware systems like PLFS [27] and ADIOS [28] cannot be applied without

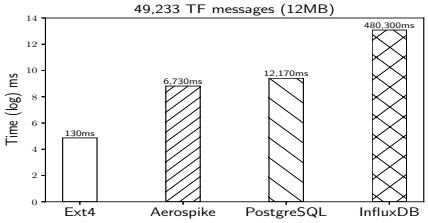

Fig. 2: A performance comparison in message insertion.

deep modification to optimize ROSbag. We conducted a group of experiments to compare PLFS with Ext4 and XFS. It shows that PLFS takes 2x longer time to write a 3.9GB bag file, and spends 1x longer time to retrieve a topic of data (Fig. 3). ADIOS requires substantial changes in the application code, which becomes a burden for a ROS application developer.

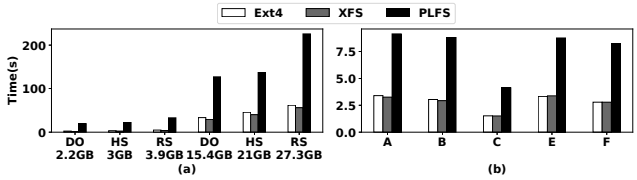

Fig. 3: (a) bag write (notions refer to Table II); (b) bag read (bag size: 2.9GB, notions refer to Table III).

The `rosbag` tool builds an index for all messages during an open operation to support data acquisition. For example, the command `bag.read_messages(topics=['foo1', 'foo2'])` can read out messages of two different topics (i.e., `foo1` and `foo2`) from a bag. Fig. 4a demonstrates the steps within a traditional bag open operation using the default `rogbag` API. There is a time-consuming iteration after the chunk info is read as `rosbag` traverses the chunk info list to locate the corresponding position of the index and build a hash table for message queries. The open operation takes $O(N)$ of time, where N is the number of entities in the chunk info list. In our experimental studies, we found that opening a 21GB bag took more than seven seconds. Note that the experiment was carried out on an SSD (solid state drive). The implication is that simply replacing a slow storage device like HDD (hard disk drive) with a high-performance SSD cannot solve the problem of inefficient bag data retrieval.

Motivation 2: there is a great need to develop a simplified indexing mechanism that can improve the efficacy of both bag manipulation and multi-queries. A sample multi-query could be `bag.read_messages(topics, start_time, end_time)`, where `start_time` and `end_time` defines a time window for querying messages of different topics.

The two motivations drive us to develop a bag optimizer for robotic analysis (BORA), a file system middleware that can optimize the storage and data acquisition of ROS bags. The key challenge of BORA is that bags were originally designed in a log-structured way to quickly record sensor data in chronological order. As a result, retrieving wanted data in a bag requires scanning the entire bag to locate the scattered messages. When the size of a bag file increases, the cost of this inefficient data retrieval becomes proportionally high. The poor performance of data retrieval of bags becomes increasingly inadequate to meet the requirements of a ROS application with some advanced data analysis needs. Another challenge is how to minimize the modifications of the existing ROS software stack shown in Fig. 1a while achieving the goal of this research. The design of BORA is presented in the next section.

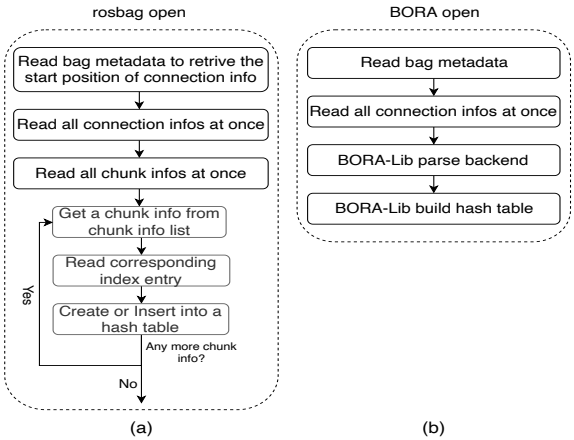

Fig. 4: (a) `rosbag` open; (b) BORA-assisted open.

III. The Design of BORA

A. Design Goals

First of all, BORA should keep the existing bag format to retain the advantages of quick high-volume data collecting and data sharing. In addition, keeping the bag format also ensures the compatibility of ROS applications. Second, BORA should provide transparency for developers so that the upper-level applications do not need to modify their interface codes. Third, BORA aims at the enhancements of bag data acquisition.

As shown in Fig. 4b, BORA reduces bag open time by eliminating the iteration existed in a traditional bag open operation shown in Fig. 4a. It utilizes a coarse-grain time indexing technique to provide a fixed time window for the messages. Using a simple calculation, BORA targets the messages within certain time slots to provide a reduced number of messages for later fine-grain looking up based on timestamps.

B. BORA Architecture

Fig. 5a illustrates the architecture of BORA. BORA sits between the ROS framework and an underlying file system. It supports the ordinary ROS-Lib with the help of the FUSE interface to retain traditional usage of bags including replay and computation graphs. We also developed a BORA-Lib, which is integrated into the ROS-Lib to achieve better performance of `rebagging` and complex data acquisition operations for robotic data analysis. To provide transparency, BORA uses a container structure so that developers can use the traditional

bag abstraction without any knowledge of BORA via a front-end path. Real data is stored as a directory via a back-end path on the underlying file system (see Fig. 5a). The container contains a data organizer and a tag manger.

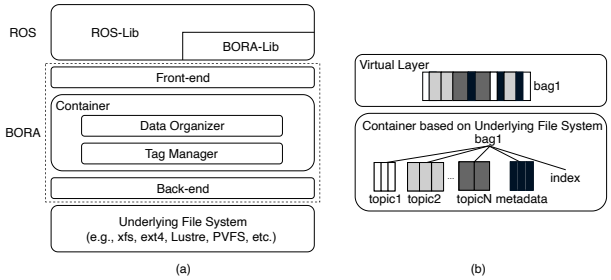

Fig. 5: (a) BORA architecture; (b) container architecture.

Container: BORA creates a container structure on the underlying file system for each logical bag file generated. Inspired by the subfiling mechanism proposed by the authors of PLFS [27], BORA splits an existing bag file into multiple single-topic files in the container to improve performance. Different from PLFS, which was designed for performance improvement in checkpoint-restart use cases, BORA focuses on offering better data query performance based on robotic data semantics. Internally, the basic structure of a container is a hierarchical directory tree consisting of a single top-level directory and multiple sub-directories. BORA creates a logical view of a single file from this container structure. Fig. 5b demonstrates the organization of the container. After a file called *bag1* is created BORA builds a container structure on the underlying file system. The container is comprised of a root directory with the same name called *bag1* and multiple sub-directories to store message data. The names of sub-directories are determined by the types of messages that are defined by topics. For each write, BORA appends the message data to the corresponding topic file and updates an entry in the index file. The index entry contains the timestamp of the write, its logical offset, its length, and a pointer to its physical location. The metadata shown in Fig. 5b is generated internally by the `rosbag` tool.

Data Organizer: Once an original bag file arrives, the data organizer reads the metadata of the bag to locate connection info records, and then reads all of them at once to identify the types of messages (i.e., topics). Next, it scans and distributes messages to distinct target sub-directories in the container structure under the specific topic names. To de-serialize the I/O operation, BORA uses one thread to scan the file and a few other threads to distribute messages to the underlying file system. The number of threads is determined by system specs.

Tag Manager: The tag manager maintains a hash table that maps types of topics to their logical locations on the underlying file system. The key of an entry in the table is a topic name and the value is its back-end path. In this way, BORA can quickly locate the path to the requested topics such as `/bag1/topic1` and `/bag1/topic2` for `rosbag` command `bag.read_messages (topics =['topic1', 'topic2'])`. The operations of BORA are detailed in Section III-C. BORA does not store the hash table but builds it whenever a bag is opened. We conducted a group of experiments to measure the overhead of building the hash table on-the-fly. When the number of topics is less than 100,000 there is no significant time difference between reading the hash table and building it on-the-fly (see Table I). The experimental results prove that the hash table construction time is determined by the number of topics (a.k.a the number of sensors that a robot is equipped). One can find that the table construction times are restrained in a range from less than one millisecond to three dozens of millisecond as the number of topics increases from 10 to 100,000. This time cost is insignificant compared to the data query time (more than dozens of seconds). Since a single robot usually is not capable of installing one hundred thousand sensors, so we can safely predict that the cost of on-the-fly hash table construction is negligible.

TABLE I: Time and Space Costs to Construct the Hash Table

Number of Topics	Hash Table Size (KB)	Time Costs(ms)
10	0.11	0.163
100	1.2	0.476
1,000	13	3.949
10,000	136	29.883
100,000	1,500	35.840

C. BORA Operations

BORA supports two ways of *rosbag* I/O operations. Common operations including `open`, `read`, and `write` are passed through ROS-Lib via the FUSE layer. In this section, we discuss three advanced operations of BORA: data duplication (copy from external storage devices), data acquisition (data query by topics), and combined data query with topics and start-end times. These operations are intercepted by BORA-Lib and then manipulated by BORA Containers (see Fig. 5a). Since all three BORA operations are performed on a bag file that has been created by the `rosbag` tool, they use bags in an offline way (see Section II-A). BORA could be integrated into a file system running on a robot so that it can manipulate bag data (e.g., storing a bag, extracting data from a bag, etc.) in an online way. However, online usage of BORA requires modifications the way of storing a bag file in the ROS framework. This may demand the support of ROS communities to promote framework updates with BORA.

BORA data duplication: ROS uses bags mainly to store and share a large volume of robotic data for later analysis. Unlike online bag writes, a bag to be duplicated has complete information of a robotic application including metadata, message data, and descriptive semantics. BORA reorganizes data arrangement when a developer copies bags from external portable storage devices or downloads them remotely. During a duplication operation, BORA creates a container structure on the back-end directory with a name same as that of the bag. And then it creates sub-directories whose names are determined by the names of topics, which can be identified

by reading the connection info from the metadata of the bag. Next, it scans messages to identify the topic that the message belongs to and then appends each message to a sub-directory with the same name. Finally, it updates the records on the index file. In this way, BORA re-distributes data to target sub-directories by scanning the file once.

Fig. 6 illustrates how BORA carries out a data duplication operation step-by-step: (1) BORA intercepts I/O requests from ROS; (2) data organizer scans and separates data to different topics; (3) data organizer distributes data by topics to a thread pool; (4) task manager assigns available threads to write data with topic to the underlying file system. Data organizer is only involved during the first time that a bag is duplicated into a BORA structure. For later data sharing, bags will be copied as sub-directory trees if a target machine installs BORA. Otherwise, they will be copied as the ordinary structures (a.k.a. "bag is a file") thanks to the container structure.

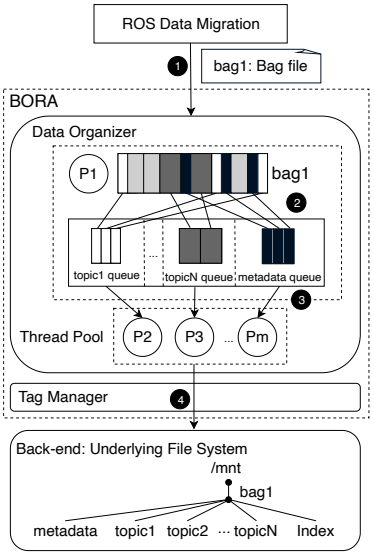

Fig. 6: BORA data duplication.

BORA data acquisition:

As aforementioned, bag is originally designed to replay the data that is collected by robots in chronological order. The replay operation can be viewed as a sequence of reads. When it comes to data acquisition for queries, the `rosbag` tool needs to iterate the bag file to build an index structure for message searching, which takes $O(N)$ time (see Fig. 4a). In order to respond to the command `bag.read_messages(topics=['topic1', 'topic2'])`, `rosbag` launches a sequence of searches to find and then pass all offsets of messages that belong to these two topics to the underlying file system. BORA enhances data acquisition operation in two steps. First, it eliminates the labor of iterations when opening a bag file (see Fig. 4b). BORA quickly parses the sub-directories of a bag on the back-end and then builds a hash table for the tag manager. In the hash table, the keys are topic names while the values are the corresponding paths. Next, BORA uses 'topic1' and 'topic2' to lookup the hash table to obtain their corresponding back-end paths, and then, passes them to the underlying file system. The underlying file system treats 'topic1' and 'topic2' data as two independent files because BORA already processed the files into large blocks of contiguous data during the data migration step. Fig. 7 shows the detailed steps in a BORA data acquisition operation: (1) BORA intercepts data queries with topic names from ROS; (2) tag manager uses topic names to find and pass their back-end paths accordingly to the underlying file system; (3) the underlying file system returns the requested topics to ROS. Note that multiple levels of parallelism in a file system can be exploited to further improve I/O performance.

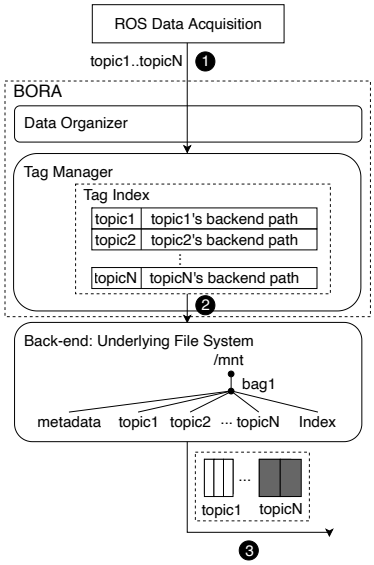

Fig. 7: BORA data acquisition.

BORA data query with start-end time: Data acquisition is a basic bag data query operation, which only uses one parameter (i.e., topic name) for looking-up messages. ROS data analysis also requires advanced data queries that combine topic names and a start-end time range when messages are collected (i.e., finding out messages of certain topics that are recorded from a start time to an end time). The existing approach first searches and distills the requested topics. Next, it performs a merge-sort of timestamps of all the messages that belong to the distilled topics to build an index entry list, which consists of timestamp and messages offsets. The time complexity of this operation is $O(NlogN)$, where N is the number of messages. The *rosbag* tool then uses topic names and start-end times to find the target messages. BORA, instead, applies a coarse-grain time indexing technique that uses a fixed time window to manage messages of each topic. Under each sub-directory, there is a priority queue that stores a pair of key-value where the key is the start time of a time window and the value is an offset list of messages that have timestamps within the time window.

The impact of the sort on data query with start-end time is two-folded: 1. it further improves the data query performance

by up to 11x for single-topic queries and up to 4x for multiple-topic queries, especially on the SSD server (see Section IV.B); 2. it provides a two-dimensional (by topics and by time) query capability based on a file system.

Fig. 8 illustrates a sample internal data structure for time indexing where the time window is set to 5 time units. For example, a key value pair named `(31, [offset list])` on the back-end directory named `/mnt/bag1/topic1/` indicates that the [offset list] holds the offsets of all topic1 messages that are collected within the start-end time range from 31 to 36 (see Fig. 8). Once a pair of start-end times is provided, BORA performs arithmetic calculations as $\lfloor start\_time/time\_window \rfloor$ and $\lceil end\_time/time\_window \rceil$ to figure out the start and end time slots of the sub-directories. Note that the value of the time window can be configured by a developer. In this way, BORA diminishes time cost of data queries by (1) reducing the number of messages that `rosbag` needs to scan to build an index entry list via a coarse-grain time index structure; (2) reducing the redundant labor for building an index entry list of the distilled messages via merge-sort.

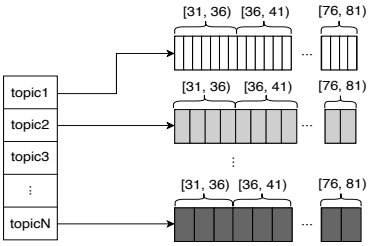

Fig. 8: The internal data structure for time indexing.

IV. BORA Evaluation

A. Experimental Setup

Very often bag files collected by an institute are not only for its own research but also for sharing with the entire robotic community. Thus, they need to be copied from robots' onboard storage devices to local servers before a further data analysis or sharing can be performed. In this scenario, BORA sits between the robots and local servers to reorganize data. A local server could be a workstation, a small cluster, or a brawnier system. In order to comprehensively evaluate the efficacy of BORA, we integrate it into three computing platforms with different scales: a single-node server, a four-node PVFS cluster, and a Tianhe-1A storage subsystem. The single-node server represents a mainstream laptop, desktop, or server that is widely used by ROS developers. The four-node PVFS cluster is employed to evaluate the performance of BORA in parallelism and scalability. Finally, the Tianhe-1A storage subsystem is utilized to evaluate the effectiveness of BORA on a real-world computing platform for data analysis of a robotic swarm [29].

All experiments are carried out under real-world applications that were collected by the Technical University of Munich [26]. Generally, a ROS application generates random access patterns due to its mixed types of sensors, different messages acquisition intervals, and distinct message sizes. However, BORA reorganizes a bag file according to the topics of messages, and then, transforms random accesses into partial sequential accesses. Table II presents the composition of a 2.9 GB Handheld SLAM bag [26]. The bag consists of seven topics and more than 98% of its data is image data (a.k.a unstructured data). Note that the unstructured data interleaves with some structured data such as IMU (nertial measurement unit), TF (transform stamped info), and Marker Arrays (arbitrary primitive shapes info). This example illustrates the importance of data acquisition optimization, especially when unstructured data is mixed with structured data.

The four real-world applications are summarized in Table III. There are two types of SLAM algorithms: handheld SLAM (HS) and robot SLAM (RS). While the former only uses RGB images and depth images, the latter employs additional data including IMU. The dynamic object (DO) is a deep learning object detection application that requires environmental semantics including TF, camera pose info, and marker array in addition to RGB images. This application demonstrates a data acquisition scenario where numerous small-size structured data and large-size unstructured data are mixed. The pre-analysis algorithms (PA) represent a common scenario where multiple stages of data analysis and modeling are required. These algorithms first pick some types of topics to build a training model and then select a few other types for subsequent phases of data analysis until final decisions are made. The entire procedure involves multiple data semantic acquisitions, multiple stages of semantic analysis of diverse algorithms, and multi-dimensional data reconstructions. Each of these stages may require a different set of topics, which leads to a varied number of topic retrievals.

B. Evaluation on Bag Duplication

Although FUSE introduces some one-time overhead, we implemented BORA with it to avoid additional API code changes. We evaluate BORA by copying one bag from an existing directory to a predefined BORA front-end on an SSD. We find that in the worst case BORA-assisted Ext4 (i.e., BORA on Ext4) is 50% slower than Ext4 and BORA-assisted XFS (i.e., BORA on XFS) is 90% slower than XFS (see Fig. 9). On average, the initial capture overhead of BORA is 26% and 51% compared to Ext4 and XFS, respectively. We further notice that as the file size increases this overhead of BORA becomes less significant. For example, when the file size is larger than 3.9 GB BORA only slows down the performance of bag write by 10% and 22% for Ext4 and XFS, respectively. Besides, we find that if a destination path is a BORA-supported one (i.e., 'BORA to BORA on Ext4' or 'BORA to BORA on XFS' shown in Fig. 9) BORA can copy a bag as fast as Ext4 and XFS. This is mainly because the data organizer of BORA is a one-time attempt to reorganize a bag into a BORA structure.

TABLE II: Data organization of a 2.9 GB bag

Id	Topic name	Type description	# of Messages	Data size
A	/camera/depth/image	Depth Image	1,429	1.64 GB
B	/camera/rgb/image_color	RGB Image	1,431	1.23 GB
C	/camera/rgb/camera_info	RGB CameraPose Info	1,432	594 KB
D	/camera/depth/camera_info	Depth CameraPose Info	1,430	594 KB
E	/cortex_marker_array	Primitive Shapes (MarkerArray)	14,487	8.4 MB
F	/imu	Inertial Measurement Unit Info (IMU)	24,367	8.4 MB
G	/tf	Transform Stamped Message (TF)	16,411	3.6 MB

TABLE III: Required Topics in Each Real-world Application

Application	Required Topics
Handheld SLAM (HS)	Depth Image, RGB Image
Robot SLAM (RS)	Depth Image, RGB Image, IMU
Dynamic Object (DO)	TF, RGB Image CameraPose, MarkerArray
Pre-analysis Algorithms(PA)	Randomly Pick

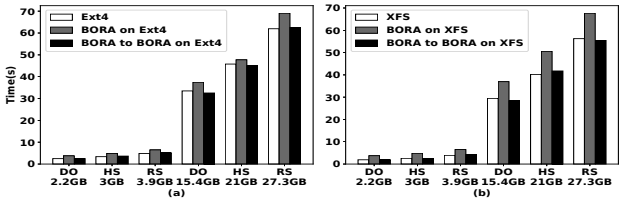

Fig. 9: Comparisons of write time of bags with distinct sizes.

C. Evaluation on A Single-Node Server

We evaluate BORA on a single-node server that equips with an Intel®Xeon®CPU E5-2603 v4 @1.70GHz, 16GB DRAM, and two 256GB NVMe SSDs. The operating system is CentOS release 6.10 (Final). We evaluate the performance of BORA in terms of major usage patterns including querying by topic and querying by start-end time.

Performance of query by topic: Fig. 10 presents the time comparisons of query by topic between two ordinary local file systems (i.e., XFS and Ext4) with and without BORA by a single topic from Handheld SLAM bags with different sizes. While the y axis represents query time, the x axis shows five topics (i.e., A, B, C, E, F shown in Table II) of Handheld SLAM bags. Fig. 10a shows the results when the bag size is 2.9 GB. We observe that on average BORA enhances 50% of query performance compared to the control groups. Especially, BORA achieves a 5x faster performance when querying a topic with a small data size (i.e., topic C in all sub-figures of Fig. 10). We further notice that BORA takes much less time to open a bag. This is because the ordinary `open` consumes more time as `rogbag` has to traverse the entire bag to generate index data during an open operation. The time consumed by `open` becomes significant when ROS only reads a small portion of data. On the contrary, this time is negligible in BORA as it only loads tag index, which is a small hash table. Comparing Fig. 10a with the rest three sub-figures, one can see that BORA consistently improves performance when the size of a bag increases from 2.9 GB to 20.3 GB.

Fig. 11 and Fig. 12 present query time comparisons of the four real-world applications between two underlying file systems (i.e., XFS and Ext4) with and without BORA with small and large bags. We observe that compared to the control groups on average BORA enhances the query performance by more than 70% and 50% for the 2.9 GB case and 21 GB case, respectively. Fig. 11 and Fig. 12 demonstrate that BORA can substantially improve query performance for all four real-world applications across all cases.

Performance of query by start-end time and topic: In this group of experiments, we evaluate BORA in terms of advanced data queries by start-end times as well as topics. Fig. 13 and Fig. 14 compare the query times under different time intervals. We fix the start time and choose an end time by adding a stair-step time interval (i.e., 5 seconds in our experiments). We can see that BORA outperforms the control groups by up to 11x in single-topic queries (see Fig. 13) and 3.5x in multiple-topic queries (see Fig. 14). This is mainly because BORA reduces the search range. When the end time keeps increasing and covers the entire bag file, BORA can still achieve around a 2x performance improvement. In particular, we find that BORA gains 11x performance improvement when querying *camera_info* data (see Fig. 13d). This is because the `rosbag` tool spends unavoidable efforts on building an index structure of the complete data set for time query even the requested data is very small. BORA can reduce indexing and searching time by providing a smaller range of data via a coarse-grained time index.

D. Evaluation on A Small Cluster

This group of experiments is conducted on a 4-node all-SSD PVFS cluster, which is interconnected with 10 Gbit/s Ethernet. Each node is equipped with one Intel®Xeon®CPU E5-2603 v4 @1.70GHz, 16 GB DRAM, and two 256 GB NVMe SSDs, which are organized as a soft RAID-0 array.

Fig. 15 presents the data query time comparisons between PVFS with and without BORA. Fig. 15a and Fig. 15b provide query times with a single topic from Handheld SLAM. We can see that BORA achieves up to 2x speedup compared to the ordinary PVFS mainly due to its negligible open time. Furthermore, we observe that BORA gains 30x speedup for querying some specific structured data such as */camera/rgb/camera_info*. This large performance improvement is attributed to BORA's extremely low-time cost of opening a bag file. Fig. 15c and Fig. 15d show query times of four real-world applications. We observe that BORA gains 2x speedup on average compared to the ordinary PVFS, which demonstrates good scalability of BORA. In addition, Fig. 16

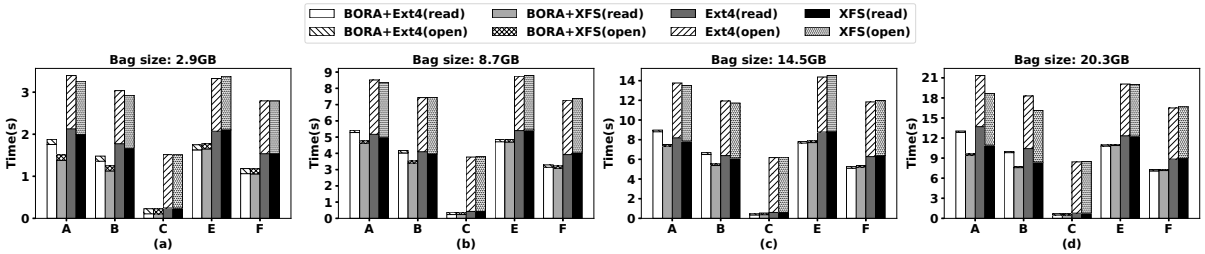

Fig. 10: Comparisons of query time by topics of Handheld SLAM on a single-node server with varied bag size.

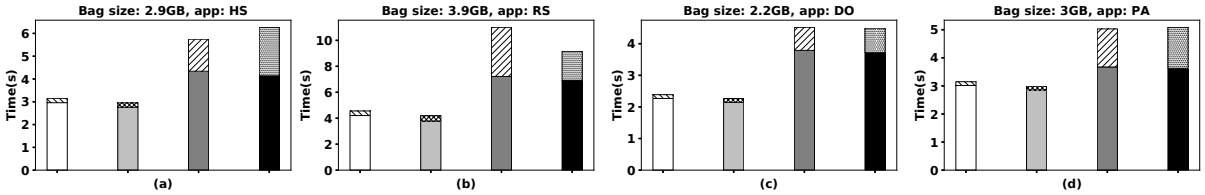

Fig. 11: Comparisons of query time by topics of real-world applications with small bags on a single-node server: (a) Handheld SLAM; (b) Robot SLAM; (c) Dynamic Object; (d) Pre-analysis Algorithms.

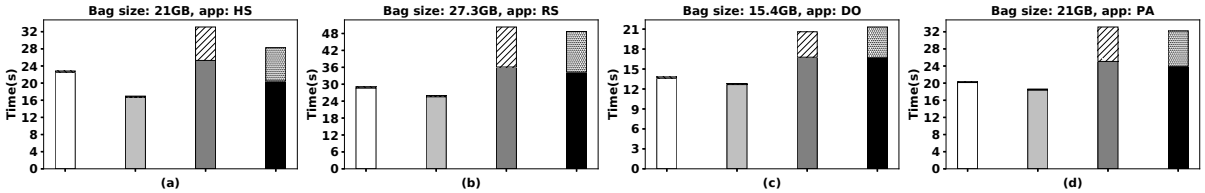

Fig. 12: Comparisons of query time by topics of real-world applications with large bags on a single-node server: (a) Handheld SLAM; (b) Robot SLAM; (c) Dynamic Object; (d) Pre-analysis Algorithms.

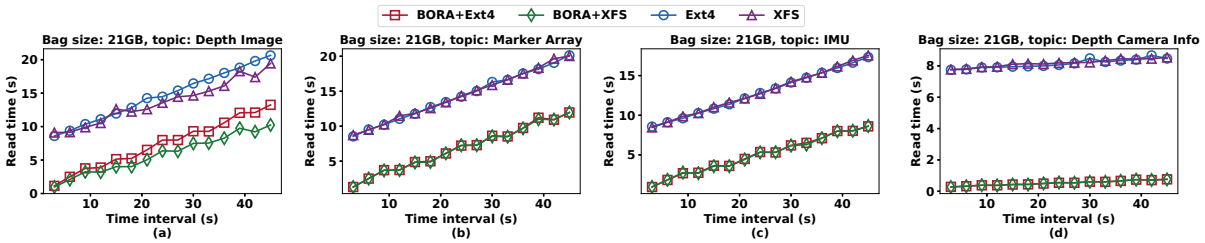

Fig. 13: Comparisons of query time by one topic and start-end time of Handheld SLAM 21 GB bag on a single-node server.

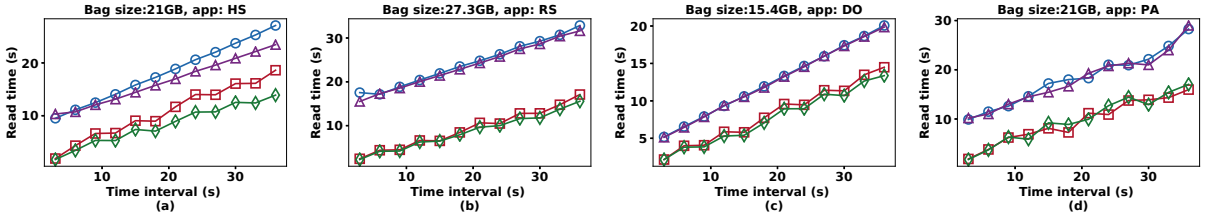

Fig. 14: Comparisons of query time by topics and start-end time of real-world applications on a single-node server: (a) Handheld SLAM; (b) Robot SLAM; (c) Dynamic Object; (d) Pre-analysis Algorithms.

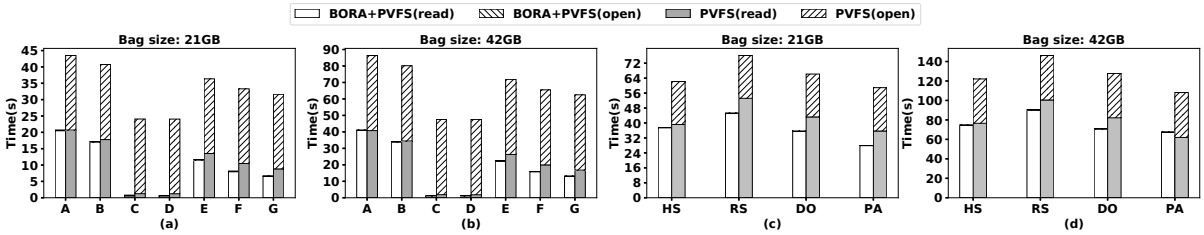

Fig. 15: Comparisons of query time by topics on a PVFS cluster: (a), (b) Handheld SLAM; (c), (d) real-world applications.

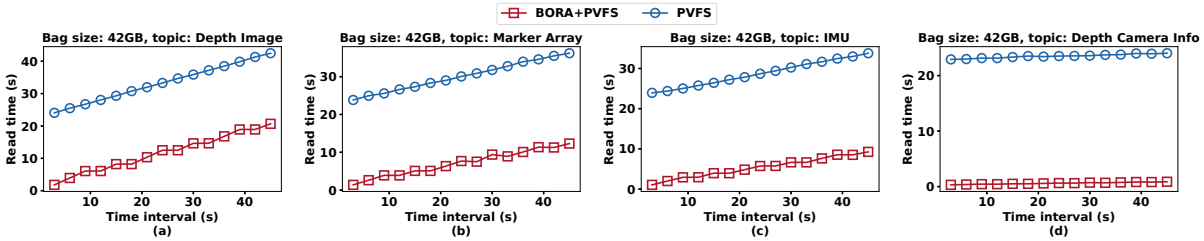

Fig. 16: Comparisons of query time by one topic and start-end time of Handheld SLAM with 42 GB bag on a PVFS cluster.

provides query performance with topics and start-end times. BORA outperforms the control group in every testing case, which proves the effectiveness of the proposed coarse-grain time indexing technique. One can see that the evaluation results on the PVFS cluster are not as good as that of on the single-node server. This is mainly due to the performance bottleneck of the network (10 Gbit/s Ethernet) and limited user-level parallelism.

E. Evaluation on A Tianhe-1A Storage Subsystem

In this sub-section, we evaluate BORA on a Tianhe-1A storage subsystem that is capable of performing data analysis of a robotic swarm [30]. A Lustre parallel file system is running on top of the storage subsystem, which consists of twelve compute nodes, three object storage servers (OSS), and four meta data servers (MDS). All of them are connected by Mellanox®Inifiband switch MT27500 ConnectX-3 (56 Gpbs). Each compute node is equipped with two Intel®Xeon®CPUs Gold 6134@3.2GHz and 384 GB DRAM. Each OSS server has two Intel®Xeon®CPUs E5-2660 v3 @2.6GHz and 128 GB DRAM. Each MDS server consists of two Intel®Xeon®CPUs E5-2650 v2 @2.6GHz and 64 GB DRAM. The total storage capacity of the cluster is 804 TB. Data analysis of a robotic swarm normally requires multiple processes to query the same topic from multiple bags simultaneously. In this scenario, a developer is retrieving messages of the same topic type (e.g., RGB Image in Table II) from multiple bags, which are collected by a large number of robots in a swarm from the same relative position concurrently. By doing so, the developer can build an object that has a multi-angle view (e.g., Bullet Time®effect in the movie The Matrix).

We configure the number of robots in a swarm to be 10, 50, and 100 to represent a small swarm, a middle-sized swarm, and a large swarm, respectively. The total number of bags in each swarm is equal to its number of robots. The storage subsystem can support up to 192 concurrent processes. In order to avoid contention, we launch up to 100 processes to open bags with one process dedicated for a particular bag. In this group of experiments, two bag sizes are employed: 21 GB and 42 GB. The processes launched open all the bags at the same time and then run the Robot SLAM application to extract data of various types including Depth Image, RGB Image, and IMU. Fig. 17 shows that the BORA-assisted Lustre outperforms the control group in all cases. More importantly, it exhibits better scalability as Fig. 17b demonstrates more than 10x overall performance improvement when totally 4.2 TB data (i.e., 100 bags with each having 42 GB data) is processed. The `read` performance gains of BORA stem from its capability of aggregating data of same topics, which provides a sequential access pattern to underlying HDDs. The significant time cost improvement (i.e., up to 3,113x in opening 100 bags with each having 42 GB data) comes from `open` operation, which is achieved by BORA's light-weight open procedure. Recall that BORA only loads a small hash table of tag index information instead of traversing the entire bag file. The substantial performance improvement in `open` shows that BORA can fully exhibit its potential when the number of bags and data volume of each bag increase. As shown in Fig. 18, BORA's coarse-grain time indexing mechanism reduces time costs by up to 4x for queries by topics and a time range.

V. Discussions

In this section, we summarize three lessons that we learned from this research.

An efficient data indexing mechanism is critical to the performance of robotic analysis: Data indexing plays a key role in achieving an efficient ROS data analysis, especially when the amount of data is large. To obtain a good performance in write, ROS spreads index information throughout a bag file. However, this index layout makes data analyzing,

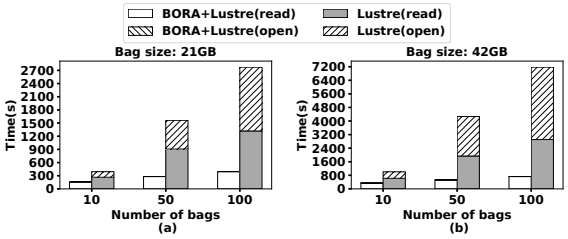

Fig. 17: Comparisons of query time of a robotic swarm on a Tianhe-1A storage subsystem: (a) 21 GB each bag; (b) 42 GB each bag.

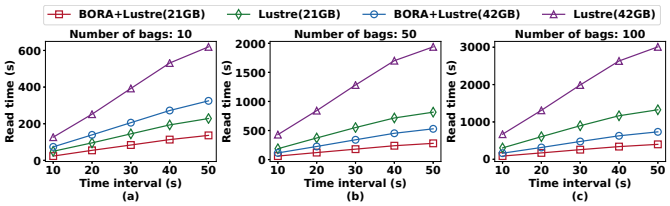

Fig. 18: Comparisons of query time by topics and start-end times of a robotic swarm on a Tianhe-1A storage subsystem.

collecting, and aggregating time-consuming. Furthermore, we observe that most data is written only once but will be read multiple times the latter. Therefore, it is essential to extract index information to a specific location, which can be quickly retrieved later. BORA can greatly boost the performance of data acquisition mainly because it employs an efficient data indexing mechanism.

Redundant data retrieval should be avoided as much as possible: In the robotic analysis, it is common to retrieve a complete collection of a dataset with multi-dimensional variables to the memory and then manipulate it. As multi-dimensional data collection becomes more complicated a data cleaning operation requires more temporal memory space to store some intermediate values, which leads to a longer data processing time. Therefore, a technique that can dynamically provide an appropriate amount of data to avoid redundant data retrieval is much needed. BORA can judiciously retrieve only desired data from a complete dataset so that redundant data retrieval is avoided, which also contributes to its high performance in data acquisition.

The potential of SSDs cannot fully exhibit without a data organization optimization in ROS software layers: Modern storage devices like SSDs can offer much higher performance than that of HDDs. However, we found that simply replacing HDDs with SSDs cannot achieve a high data acquisition performance in ROS applications. The reason is that when high-bandwidth SSDs are employed software latency becomes a dominant factor that determines the performance of data processing. Thus, optimizing ROS data organization in various software layers becomes a must.

VI. Related Work

To the best of our knowledge, this research is the first study that focuses on optimizing data organization of ROS bags for robotic analysis. Therefore, little closely related work is found in the literature. In this section, we first briefly summarize some widely used I/O middleware systems. Next, we introduce prior investigations in DMBS for ROS applications.

Although several adaptive I/O middleware systems [31] [28] [27] [32] have been developed to improve the I/O efficiency for various applications, none of them can work with the ROS framework. HDF5 is a data model containing complex data objects and metadata [31]. Its file format is incompatible with that of bag. ADIOS [28] provides a high-level I/O API to perform aggressive data relocation within the checkpoint, but it requires application modifications. Unlike ADIOS, PLFS [27] uses a user-level file system interface to gain performance benefits without modifying any application. ROMIO [32] is an MPI-IO implementation, which delivers high performance I/O in the presence of noncontiguous requests. It requires high-level libraries like MPICH and significant code modification if it is employed in the ROS framework. Like PLFS, BORA requires no code modification in any file system or application. PLFS [27] might be the most similar in philosophy to BORA, particularly in that both employing a container structure to provide a transparent data management layer. Table IV summarizes the differences between these middleware systems.

TABLE IV: I/O Middleware System Comparison

	Interposition	Usage	App. Modification
HDF5	Library	Scientific Data	No
ADIOS	Library	Checkpoint-restart	No
PLFS	FUSE or Library	Checkpoint-restart	Yes
ROMIO	Library	MPI-IO	No
BORA	FUSE or Library	Bag Enhancement	Yes

A handful of database systems had been developed to facilitate robotic analysis from various angles. While MongoDB was developed to store and manage document data provided by a set of domestic mobile robot sensors for a smart-home environment [33], a NoSQL graph database named Neo4J was proposed to store and query long-term human-robotic interaction data for high data availability [34]. Aerospike is an in-memory NoSQL database system [23]. It faces a challenge in striking a balance among capacity, performance, and data persistence. Unlike [33] [34] that targeted a single-robot analysis platform, Cassandra was developed to store a large amount of robotic data for the progressive assignment algorithm for a multi-agent system [35]. Along the same line, PostgreSQL was built on a cloud platform to manage a range of robotic control data from six-axis robots to improve control precision with intelligent functions [24]. InfluxDB [25] and BtrDB [36] were designed to optimize the performance of data retrieval from a time series database. Unfortunately, they do not support complex array structures, and thus, are inadequate to process rich ROS data. Even though these DBMS systems were proposed to make robotic data analysis easier in some

specific scenarios, they lost the advantages of existing ROS bag mechanisms (i.e., portability and generality) as none of them kept robotic use cases in mind.

VII. Conclusions

Software development for robots is often more challenging than other types of software development. One reason is that algorithm testing can be time-consuming and error-prone. ROS separates low-level hardware control and high-level decision making into separate programs. It also provides a simple way to record and playback sensor data and other types of messages. By recording a robot's sensor data, a developer can replay it many times to test various algorithms on that data. ROS is a *de facto* robotic software development platform as it received extensive support from the robotics community.

More than just replaying messages, some emerging robotic applications [8] [9] now require the capabilities of advanced message acquisition and rich queries. How to retain the advantages of the existing ROS bag mechanism while enhancing message extracting and querying becomes a new challenge. To solve it, we develop a prototype of a file system middleware called BORA, which is then integrated into three computing platforms. Further, we evaluate BORA in terms of data queries using four real-world ROS applications. Our experimental results demonstrate that BORA can significantly improve data query performance. Besides, it meets the data analysis requirements of robot swarms.

Acknowledgment

We would like to thank the anonymous reviewers for their tremendous feedback and comments, which have substantially improved the content and presentation of this paper. We are thankful to Laurent Kneip and Sören Schwertfeger from the ShanghaiTech Automation and Robotics Center (STAR) for their invaluable feedback on this work. We are also indebted to Yong Dong from the State Key Lab of High-Performance Computing, China to assist us to run tests on the Tianhe-1A supercomputer system. Shu is deeply thankful to Hong Jiang and Adam Manzanares for their contiguous encouragement. Shu Yin's research is supported by the China Postdoctoral Science Foundation under Grant 2015M572708, and ShanghaiTech University under a start-up grant. Tao Xie's work is partially supported by the US National Science Foundation under grant CNS-1813485.

References

[1] I. Afanasyev, A. Sagitov, and E. Magid, "Ros-based slam for a gazebo-simulated mobile robot in image-based 3d model of indoor environment," in *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, 2015, pp. 273–283.

[2] A. Maldonado, U. Klank, and M. Beetz, "Robotic grasping of unmodeled objects using time-of-flight range data and finger torque information," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 2586–2591.

[3] S. Zaman, W. Slany, and G. Steinbauer, "Ros-based mapping, localization and autonomous navigation using a pioneer 3-dx robot and their relevant issues," in *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*. IEEE, 2011, pp. 1–5.

[4] Y. Hua, S. Zander, M. Bordignon, and B. Hein, "From automationml to ros: A model-driven approach for software engineering of industrial robotics using ontological reasoning," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–8.

[5] Y. Yu, X. Wang, Z. Zhong, and Y. Zhang, "Ros-based uav control using hand gesture recognition," in *2017 29th Chinese Control And Decision Conference (CCDC)*. IEEE, 2017, pp. 6795–6799.

[6] C. Hu, C. Hu, D. He, and Q. Gu, "A new ros-based hybrid architecture for heterogeneous multi-robot systems," in *The 27th Chinese Control and Decision Conference (2015 CCDC)*. IEEE, 2015, pp. 4721–4726.

[7] ROS, https://wiki.ros.org, 2019.

[8] J. A. Caley, N. R. Lawrance, and G. A. Hollinger, "Deep learning of structured environments for robot search," *Autonomous Robots*, vol. 43, no. 7, pp. 1695–1714, 2019.

[9] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, and J. J. Berlles, "Stereo parallel tracking and mapping for robot localization," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 1373–1378.

[10] T. Niemueller, G. Lakemeyer, and S. S. Srinivasa, "A generic robot database and its application in fault analysis and performance evaluation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 364–369.

[11] R. Ravichandran, E. Prassler, N. Huebel, and S. Blumenthal, "A workbench for quantitative comparison of databases in multi-robot applications," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 3744–3750.

[12] J. S. van der Veen, B. van der Waaij, and R. J. Meijer, "Sensor data storage performance: Sql or nosql, physical or virtual," in *2012 IEEE Fifth International Conference on Cloud Computing*, June 2012, pp. 431–438.

[13] Y. Koo and S. Kim, "Distributed logging system for ros-based systems," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Feb 2019, pp. 1–3.

[14] K.-M. Kim, C.-J. Nan, J.-W. Ha, Y.-J. Heo, and B.-T. Zhang, "Pororobot: A deep learning robot that plays video q&a games," in *2015 AAAI Fall Symposium Series*, 2015.

[15] W. Chen, T. Qu, Y. Zhou, K. Weng, G. Wang, and G. Fu, "Door recognition and deep learning algorithm for visual based robot navigation," in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. IEEE, 2014, pp. 1793–1798.

[16] D. Ribeiro, A. Mateus, P. Miraldo, and J. C. Nascimento, "A real-time deep learning pedestrian detector for robot navigation," in *2017 IEEE international conference on autonomous robot systems and competitions (ICARSC)*. IEEE, 2017, pp. 165–171.

[17] F.-Y. Wang, P. J. Lever, and B. Pu, "A robotic vision system for object identification and manipulation using synergetic pattern recognition," *Robotics and computer-integrated manufacturing*, vol. 10, no. 6, pp. 445–459, 1993.

[18] Z. Erickson, S. Chernova, and C. C. Kemp, "Semi-supervised haptic material recognition for robots using generative adversarial networks," *arXiv preprint arXiv:1707.02796*, 2017.

[19] W. Lawson, E. Bekele, and K. Sullivan, "Finding anomalies with generative adversarial networks for a patrolbot," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 12–13.

[20] Y. Xia and J. Wang, "A dual neural network for kinematic control of redundant robot manipulators," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 31, no. 1, pp. 147–154, 2001.

[21] T. Niemueller, G. Lakemeyer, and S. S. Srinivasa, "A generic robot database and its application in fault analysis and performance evaluation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 364–369.

[22] A. J. Fiannaca and J. Huang, "Benchmarking of relational and nosql databases to determine constraints for querying robot execution logs," *Computer Science & Engineering, University of Washington, USA*, pp. 1–8, 2015.

[23] V. Srinivasan, B. Bulkowski, W.-L. Chu, S. Sayyaparaju, A. Gooding, R. Iyer, A. Shinde, and T. Lopatic, "Aerospike: Architecture of a real-time operational dbms," *Proc. VLDB Endow.*, vol. 9, no. 13, p. 1389–1400, Sep. 2016. [Online]. Available: https://doi.org/10.14778/3007263.3007276

[24] W. Chang, S. Lin, J. Hsu, and B. Hsu, "Automatic path planning of robot for intelligent manufacturing based on network remoted controlling and

simulation," in *2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, July 2019, pp. 164–168.
[25] S. N. Z. Naqvi, S. Yfantidou, and E. Zimányi, "Time series databases and influxdb," *Studienarbeit, Université Libre de Bruxelles*, 2017.
[26] T. U. of Munich, https://vision.in.tum.de/data/datasets/rgbd-dataset, 2019.
[27] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "Plfs: a checkpoint filesystem for parallel applications," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–12.
[28] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible io and integration for scientific codes through the adaptable io system (adios)," in *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, ser. CLADE '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 15–24. [Online]. Available: https://doi.org/10.1145/1383529.1383533
[29] S. Kornienko, O. Kornienko, and P. Levi, "Generation of desired emergent behavior in swarm of micro-robots," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ser. ECAI'04. NLD: IOS Press, 2004, p. 239–243.
[30] M. Patil, T. Abukhalil, S. Patel, and T. Sobh, "Ub robot swarm: Design, implementation, and power management," in *12th IEEE International Conference on Control and Automation (ICCA)*, June 2016, pp. 577–582.
[31] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the hdf5 technology suite and its applications," 03 2011, pp. 36–47.
[32] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective i/o in romio," in *Proceedings - Frontiers 1999, 7th Symposium on the Frontiers of Massively Parallel Computation*, United States, Jan. 1999, 7th Symposium on the Frontiers of Massively Parallel Computation, Frontiers 1999 ; Conference date: 21-02-1999 Through 25-02-1999.
[33] N. Bellotto, M. Fernández Carmona, and S. Cosar, "ENRICHME integration of ambient intelligence and robotics for AAL," in *Wellbeing AI: From Machine Learning to Subjectivity Oriented Computing (AAAI 2017 Spring Symposium)*, March 2017.
[34] N. Koster, S. Wrede, and P. Cimiano, "A model driven approach for eased knowledge storage and retrieval in interactive hri systems," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, Jan 2018, pp. 113–120.
[35] C. McCord, J. P. Queralta, T. N. Gia, and T. Westerlund, "Distributed progressive formation control for multi-agent systems: 2d and 3d deployment of uavs in ros/gazebo with rotors," in *2019 European Conference on Mobile Robots (ECMR)*, Sep. 2019, pp. 1–6.
[36] M. P. Andersen and D. E. Culler, "Btrdb: Optimizing storage system design for timeseries processing," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, Feb. 2016, pp. 39–52. [Online]. Available: https://www.usenix.org/conference/fast16/technical-sessions/presentation/andersen

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

0.1 Abstract

This description contains the information needed to launch all experiments of SC20 paper "BORA: A Bag Optimizer for Robotic Analysis". More precisely, we explain how to compile and run BORA used in our evaluation part. The source code of BORA is temporarily unavailable to the public because we are filing a PCT (Patent Cooperation Treaty, an international patent law treaty) patent (the patent number can be provided if it does not violate the anonymous role) for this project. The patent application already passed the International Search phase and is currently in the International Publication phase. Once it passes the International Publication phase, we will immediately disclose the BORA source code to the public.

0.2 Description

- **Program: C/C++, Python 3.7.6 and FUSE 2.9.4**
- **Run-time state: The system is idle and only running our tests**
- **Output: Sensor messages queried by the user**
- **Experiment workflow: Clone project, compile and install BORA. Install ROS framework. Specify the front-end and back-end storage directories, mount BORA to the front-end directory, then save the bag file to the front-end, and finally use ROS-Lib to operate.**
- **Publicly available?: The source code will be published as soon as possible once the patent application is accepted.**

0.2.1 Hardware dependencies. BORA has been tested on a variety of x86 machines.

0.3 Installation

(1) First you must clone BORA code to the local machine: The code will be published to github, after the source code is made public.
(2) Get into BORA directory and compile it.

```
$ cd BORA && mkdir build
$ cd build && cmake ..
$ make -j4
$ sudo make install
```

0.4 Experiment workflow

(1) Create a config file under *$HOME/.borarc*, the format is as follows.

```
- mount_point: /mnt/bora
  backends:
    - location: posix:///home/foo/ssd
```

(2) Mount BORA

```
$ ./bin/bora /mnt/bora
```

(3) Then we can put bag file to the mount point and use ROS library to use it. The method of use is the same as the ROS official document.

0.5 Evaluation and expected result

The expected results include sensor messages (GPS, IMU, Image etc.)

ARTIFACT AVAILABILITY

Software Artifact Availability: Some author-created software artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Hardware Artifact Availability: All author-created hardware artifacts are maintained in a public repository under an OSI-approved license.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

Author-Created or Modified Artifacts:

```
Persistent ID: https://github.com/SC20-bora/bora
↪ (DOI: 10.5281/zenodo.3957776)
Artifact name: The preliminary link to BORA source
↪ code. The code will be disclosed once our PCT
↪ patent passes the International Search phase.
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: We run Author-Kit (see https://github.com/SC-Tech-Program/Author-Kit) to gather the hardware environment information. First of all, We evaluate BORA on a single-node server that equips with an Intel Xeon CPU E5-2603 v4 @1.70GHz, 16GB DRAM, and two 256GB NVMe SSDs. We then run BORA on a 4-node all-SSD PVFS cluster, which is interconnected with 10 Gbit/s Ethernet. Each node is equipped with one Intel Xeon CPU E5-2603 v4 @1.70GHz, 16GB DRAM, and two 256GB NVMe SSDs, which are organized as a soft RAID-0 array. At last, we evaluate BORA on a production cluster. The production cluster is running on top of a Lustre parallel file system which consists of 12 compute nodes, three object storage servers (OSS), and four metadata servers (MDS), which are connected by Mellanox Inifiband switch MT27500 ConnectX-3 (56Gpbs). Each compute node is equipped with two Intel Xeon CPUs Gold 6134@3.2GHz and 384GB DRAM. Each OSS server has two Intel®Xeon CPUs E5-2660 v3 @2.6GHz and 128GB DRAM. Each MDS server consists of two Intel Xeon CPUs E5-2650 v2 @2.6GHz and 64GB DRAM. The total storage capacity of the cluster is 804TB.

Operating systems and versions: CentOS 6.10 Final

Compilers and versions: CMake 3.13 and gcc 7.4.0

Applications and versions: Real world robotic applications presented by Technical University of Munich

Libraries and versions: ROS Library, FUSE 2.9.4 or above

Key algorithms: The paper describes in detail how BORA organizes the bag file and how to build an index for quick query

Input datasets and versions: All experiments are carried out under real-world applications that were collected by the Technical University of Munich. (https://vision.in.tum.de/data/datasets/rgbd-dataset)