Improved Checkpoint Using the Effective Management of I/O in a Cloud Environment

Bakhta Meroufel
University of Oran1, Algeria

Ghalem Belalem
University of Es Senia Oran1, Algeria

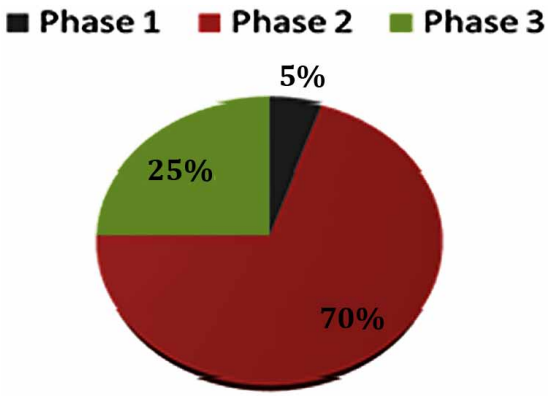
INTRODUCTION

The emergence of cloud computing has brought a new dimension to the world of information technology. Although cloud computing offers several advantages such as virtualization, cost reduction, multi-tenancy, etc., there are risks and failures associated with it (Yang *et al.*, 2014). A key challenge for research in cloud computing is to ensure the reliability of the system without reducing the overall system performance. Among of fault tolerance, there is the strategy of checkpointing. The major problem of checkpointing is the overhead caused by the storage time of checkpointing files in stable storage, this time is estimated at 70% of checkpointing process time caused by the storage (Ouyang *et al.*, 2009a;Cornwell &kongmunvattana, 2011a), Figure1 shows the main phases of the checkpointing process. This process is based on three phases: i) suspend communication between processes and ensure consistent state; ii) use the checkpointing library to create and store checkpoints; iii)re-connect processes and continue execution.

The aim of our work is to minimize the overhead of checkpointing by minimizing its storage time. To ensure this goal, we improve the I/O management and we propose a checkpointing strategy of three phases:

1. The construction of VRbIO topology (Virtual RbIO): RbIO proposed in (Lui *et al.*, 2010) is a virtual hierarchical topology; it minimizes checkpointing time and I/O time at the same time. In our system, each VM has a reactive agent responsible of the local I/O management; at the end of this phase some of these reactive agents will be activated to manage the I/O of a group of VMs of the server. In this case, the I/O will be hierarchical.
2. Creating the checkpointing files using coordinated checkpointing protocol.
3. Ensuring a lightweight and fault-tolerant storage of these files by using Collective and Selective Data Sieving input/output (CSDS I/O), which is executed by only the active agents. CSDS is an improved ROMIO I/O strategy. However, this strategy has several problems and limitations (Fu *et al.*, 2011).

Figure 1. The time of the phases of the checkpointing process



DOI: 10.4018/978-1-5225-2255-3.ch096

Our algorithm with its three phases provides solutions for most issues raised by the use of classical checkpointing with ROMIO as an I/O strategy. The rest of the chapter is organized as follows: Section 2 presents the background in the field of aggregating I/O techniques with a comparative study. ROMIO and its features are illustrated in Section 3. Section 4 presents our contribution, each service of this contribution is described in details, and all the problems cited in previous section are solved in this section. Section 5 presents some experimental results, followed by a conclusion and future research directions.

BACKGROUND

An important reason for the limitations of I/O systems is that applications often send smaller queries disjoint. This access mode generates a first additional cost to the large number of applications running on various transmission channels, but more significantly increases the processing time of the latter (Sadiku *et al.*, 2014). To deal with this problem, several "aggregation" methods have been proposed we can distinguish two types of aggregations strategies: dependent and collective.

Independent I/O is a straightforward form of I/O and is widely used in parallel applications. This form of I/O can be called independently by an individual process or any subset of processes of a parallel application. The advantage of independent I/O is that users have the freedom to perform I/O for each individual process or any subset of the processes that open the file.

The buffering is an Independent I/O (Cornwell & Kongmunvattana, 2011a). In conventional strategies, the write operation transfers data from the buffer to the local disk from their reception. Buffering proposes that the buffer will be used for temporary storage of I/O. The write operation includes small blocks in a buffer (of limited size). Once the buffer is completely filled, it will be forwarded to the local disk.

The "List I/O" approach (Thakur *et al.*, 1999a) provides routines to indicate within a single call access number. A list of torque (offset, size) describes the distribution of data in memory and a similar list is used to perform matching on disk.

Data sieving (Thakur *et al.*, 1999a) is one of the techniques proposed to address this issue by aggregating small requests into large ones. Instead of accessing each small piece of data separately, data sieving accesses a large contiguous scope of data that includes the small pieces of data. The additional unrequested data are called holes. The size of holes compared to the requested data controls the efficiency of data sieving.

For many parallel applications, even though each process may access several non-contiguous portions of a file, the requests of multiple processes are often interleaved and may constitute a large contiguous portion of a file together (Chen *et al.*, 2010). In order to achieve better I/O performance, a group of processes may cooperate with each other in reading or writing data in a collective and efficient way, which is known as collective I/O.

The collective I/O is a general idea that exploits the correlations among accesses from multiple processes of a parallel application and optimizes its I/O accesses. The basic idea behind this technique is to coordinate I/O accesses from different processors. The processors exchange information regarding what data each of them needs to access. This information is used to derive an efficient I/O schedule.

The collective I/O can distinguished by the "physical place" where the operation group is performed: if aggregation is executed among processes, the most used method is the "Two-Phase I/O" approach; if aggregation is performed at the records, we are talking about system "Disk-Directed I/O" if the approach is finally realized within a server, the method is "server-directed I/O".

"Two-Phase I/O", this method (Del Rosario *et al.*, 1993), as its name suggests, consists of two main phases: after a consensus between the