

# Can Checkpoint/Restart Mechanisms Benefit from Hierarchical Data Staging?

Raghunath Rajachandrasekar   Xiangyong Ouyang  
Xavier Besseron   Vilobh Meshram  
Dhabaleswar K. Panda

Network-Based Computing Laboratory  
Department of Computer Science and Engineering  
The Ohio State University



Resilience Workshop  
August 30, 2011

# Outline

- 1 Introduction
- 2 Hierarchical Data Staging
- 3 Experimental Results
- 4 Conclusion

# Outline

- 1 Introduction
- 2 Hierarchical Data Staging
- 3 Experimental Results
- 4 Conclusion

# Context

## Fault-Tolerance has become a necessity

- Increasing number of components in High-End Computing (HEC) systems
- Detrimental impact on Mean Time Between Failure (MTBF)

## Checkpoint/Restart approach is heavily adopted

- Heavy IO overhead
- **Underlying filesystem is the bottleneck**

# Context

## Fault-Tolerance has become a necessity

- Increasing number of components in High-End Computing (HEC) systems
- Detrimental impact on Mean Time Between Failure (MTBF)

## Checkpoint/Restart approach is heavily adopted

- Heavy IO overhead
- **Underlying filesystem is the bottleneck**

## How to reduce the checkpointing time?

We propose a Hierarchical Data Staging architecture

# Research Challenges

- How to design Hierarchical Data Staging architectures that can relieve compute nodes from checkpointing overheads?
- How to leverage high-speed networks and new storage media such as SSDs to accelerate IO performance?
- How much performance penalty should the application pay to adopt such hierarchical staging strategies?

# Background

## FileSystem in Userspace (FUSE)

- Allows for the creation of virtual filesystems at user-level
- Kernel module to perform privileged operations
- Each file operation is mapped to a user function

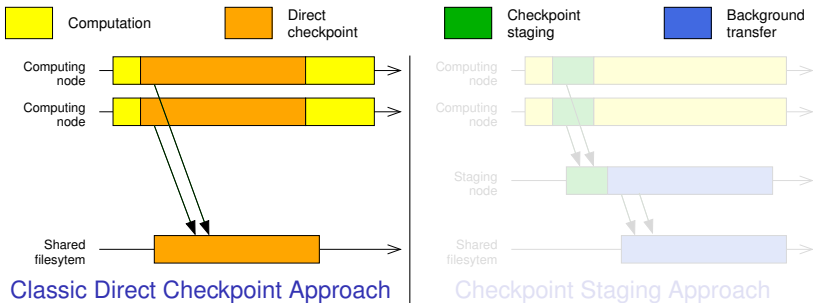
## InfiniBand and RDMA

- Open standard of high-speed interconnect
- Provides send-receive semantics
- Remote Direct Memory Access (RDMA) provides direct access to remote node's memory (the remote CPU is not used)

# Outline

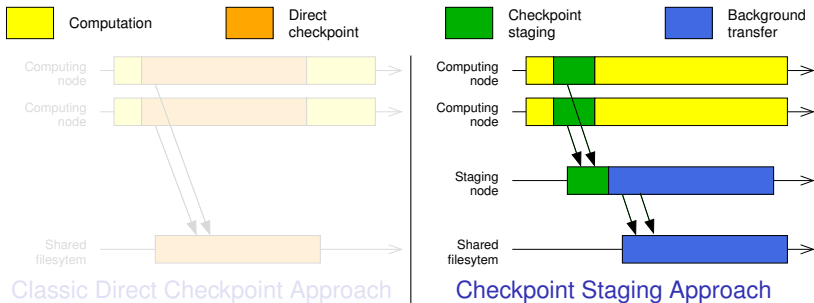
- 1 Introduction
- 2 Hierarchical Data Staging**
- 3 Experimental Results
- 4 Conclusion

# Hierarchical Data Staging Principle



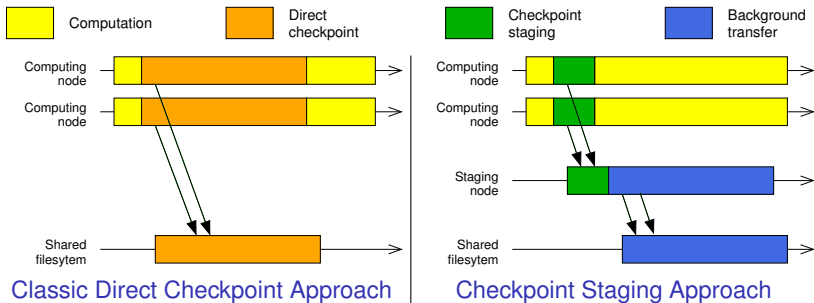
- Checkpoint files are directly written to back-end filesystem
- Contention due to multiple parallel writes
- Application is blocked until the checkpoint is written

# Hierarchical Data Staging Principle



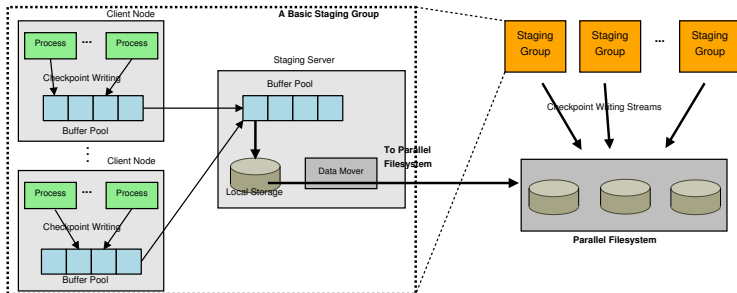
- Checkpoint files are written to staging servers
- Staging node number is proportional to computing node number
- Application resumes as soon as the data is written to the staging server
- Checkpoint files are transferred in background to the back-end filesystem

# Hierarchical Data Staging Principle



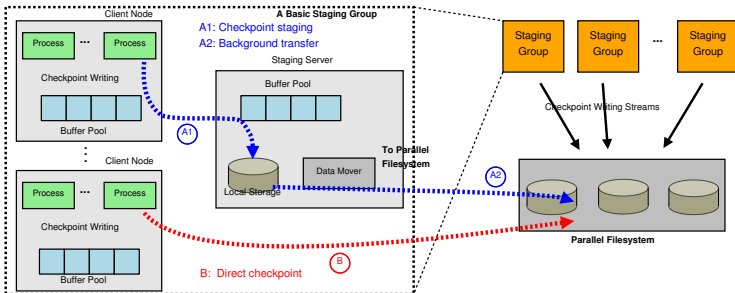
- Hierarchical transfer  $N \rightarrow S \rightarrow 1 \Rightarrow$  less contention
- Computation and data transfer are overlapped
- Staging nodes can be located close to the computing nodes
- Staging nodes can use fast storage devices (SSDs)
- Checkpoint files eventually reach the same media

# Hierarchical Data Staging Design



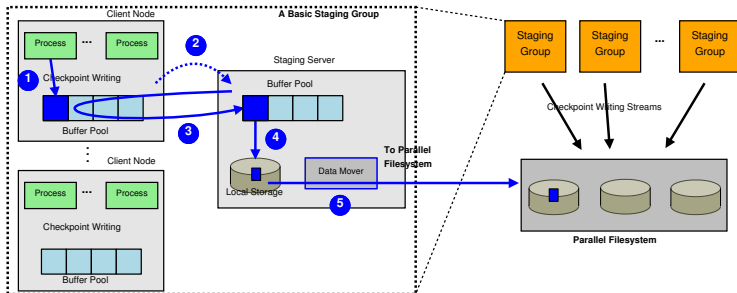
- Computing nodes are organized into staging groups
- FUSE-based staging filesystem mounted on the computing nodes
- Data transfer within a group uses RDMA
- Data is staged on the local storage and eventually written to the back-end parallel filesystem

# Hierarchical Data Staging Design



- Direct checkpoint approach in red
- Checkpoint staging approach in blue

# Hierarchical Data Staging Design



- 1 Data from write() operations are aggregated in buffers
- 2 When a buffer is full, a signal is sent to the staging server
- 3 The staging server use an RDMA read to transfer the buffer locally
- 4 Dedicated threads write data from the buffer pool to the local storage
- 5 Once all checkpoints are stored, the checkpoint files are copied on the parallel filesystem one after another

# Reliability Concerns

## After a checkpoint

- Checkpoint files are stored in the same filesystem
- Same reliability

## During the checkpoint, from the application point of view

- Checkpoint is faster
- It is better regarding failure of the computing nodes

## During a checkpoint, from the staging nodes point of view

- Staging nodes are additional points of failure
- Old checkpoint files shouldn't be deleted until all new checkpoint files are transferred to the back-end parallel filesystem

# Outline

- 1 Introduction
- 2 Hierarchical Data Staging
- 3 Experimental Results**
- 4 Conclusion

# Experimental Testbed

## 64-node InfiniBand Linux cluster

- 8 cores/node – 2 Intel Xeon 2.33Ghz CPUs
- 6GB main memory & 250GB ext3 disk /node
- Mellanox MT25208 DDR InfiniBand cards (20 Gbit/s) with OFED 1.5.1
- Linux kernel 2.6.30 with FUSE library 2.8.5

## 1 back-end parallel filesystem

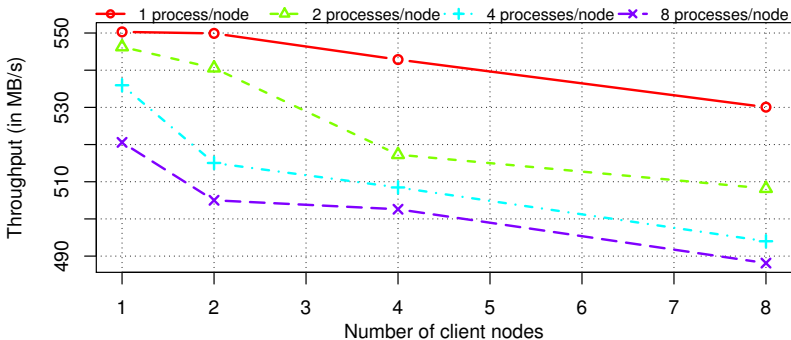
- Lustre 1.8.3 using IB transport
- 1 MDS & 1 OSS using 12-HDD RAID-0

## 4 staging nodes

- 80GB PCIExpress-based SSDs card on each
- 2 of them using Fusion-io ioXtreme cards (350 MB/s write throughput)
- 2 of them Fusion-io ioDrive cards (600 MB/s write throughput)

# Profiling of one Staging Server

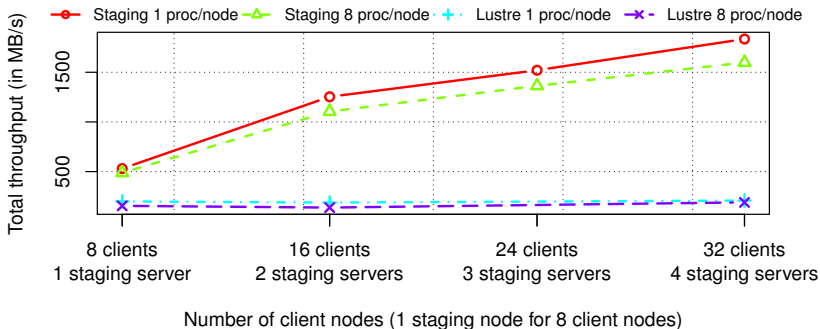
IO throughput of one staging server (IOZone benchmark)



- Maximal throughput: 550 MB/s with 1 client and 1 process
- Theoretical write throughput of SSD: 600 MB/s  
⇒ InfiniBand network is not a bottleneck
- As number of processes/node increases, there is contention at the SSD
- Throughput with 64 processes & 8 clients: 488 MB/s (only a 11% decline)

# Scalability Analysis

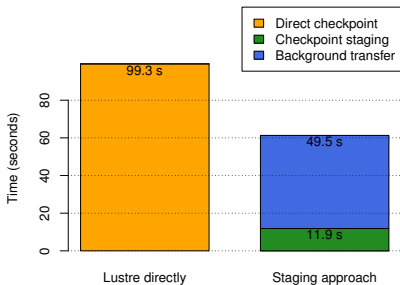
IO throughput with increasing number of staging servers (IOZone benchmark)



- Each process writes 1 GB with a 1 MB record size
- Staging architecture scales as the staging groups are increased
- Maximal aggregated throughput: 1,834 MB/s
- Theoretical aggregated write throughput of all SSDs: 1,900 MB/s

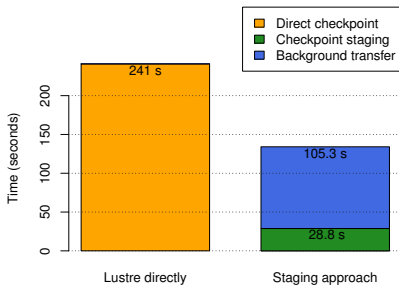
# Evaluation with Applications

Real checkpoints using MVAPICH2 1.6 and NAS benchmarks 3.3.1



LU.D.128

Checkpoint size = 13.7 GB



BT.D.144

Checkpoint size = 29.8 GB

- Background transfer time is lesser than direct checkpointing time due to reduced contention on the Lustre filesystem
- Checkpointing time, as seen by the application, is 8.3 time lesser with the staging approach

# Outline

- 1 Introduction
- 2 Hierarchical Data Staging
- 3 Experimental Results
- 4 Conclusion

## Related Works

### PLFS: a checkpoint filesystem

- Deals with scenario where all processes write to the same file

### SCR: Multi-level checkpointing

- Different checkpoint levels based on the failure type probability
- In our approach, the staging node is only a temporary storage

### Staging framework in Open MPI

- Checkpoint is staged on the local disk only

### DataStager: a scalable data staging service

- Generic service for I/O staging, also based on InfiniBand RDMA
- Our work is specialized for Checkpoint/Restart

# Conclusion

## Scalable architecture for checkpointing

- Staging nodes can be added proportionally to the number of computing nodes
- Leverage SSDs at a reduced cost

## Reduce IO contention

- Hierarchical architecture
- Computing nodes → Staging nodes → Parallel filesystem

## Overlap checkpoint write and computation

- Application can resume as soon as the checkpoint are staged
- Checkpoint is written to the parallel filesystem in background

## Future Works

### Take advantage of the CPU on the staging nodes

- On-the-fly compression to reduce the checkpoint size
- Detect duplicated data

### Study perturbations due to background transfer

- Background checkpoint transfer & application communication compete together
- Provide a QoS approach?

### Offload other checkpoint protocols

- Apply to message logging

# Thanks for your attention

## Questions ?



Network-Based Computing Laboratory

<http://nowlab.cse.ohio-state.edu/>

MVAPICH Web Page

<http://mvapich.cse.ohio-state.edu/>

## Implementation details

FUSE-based virtual staging filesystem mounted on the computing nodes

- POSIX-like semantics
- Transparent for the application
- Works with any checkpoint software

RDMA operations between the client and staging servers

- Staging servers read client buffers using RDMA read
- Use InfiniBand Verbs API
- IB buffer management taken care by the staging server

In-memory aggregation on the computing nodes

- Multiple small writes are aggregated in one large buffer

Back-end parallel filesystem is mounted on the staging nodes