Automatic Tuning of Parallel Read and Write Performance
Suren Byna

Contributors: Prabhat, John Wu, Bing Dong (LBNL), Babak Behzad and Mark Snir (UIUC), Houjun Tang, Chris Zou, Nagiza Samatova (NCSU), Quincey Koziol (The HDF Group)

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. The authors would like to thank NERSC and Cray staff for troubleshooting I/O issues on Hopper and Edison. We also thank TACC and Stampede admin staff. We would also like to thank members of the HDF Group and Burhan Loring and Oliver Rübel of the Visualization group at LBNL CRD.

Motivation

Parallel I/O
- An essential component of modern HPC
- Scientific simulations periodically write their state as checkpoint datasets
- Input and output datasets have become larger and larger
- Optimal I/O performance is critical for utilizing the power of HPC machines

Complex Parallel I/O Software Stack
- Different optimization parameters at each layer of the I/O software stack
- Complex inter-dependencies between these layers
- Highly dependent on the application, HPC platform, and problem size/concurrency
- Application developers need good I/O performance without becoming experts on the I/O

Tuning Writing Data with Model-based training
- Three Phases
  - Training phase: Using a training set, develop an empirical write performance model
  - Pruning phase: Using the model predicted write times select the top k configurations that have the best performance. Store the tuned parameters of the best performing configuration
  - Refitting phase (optional): Refit the model to obtain the current characteristics of the I/O system

Performance Modeling and Tuning Writes
Nonlinear regression model
- We consider smooth, nonlinear models, which can be written as linear combinations of \( n \) nonlinear basis functions \( \phi \).
- \( m(x;\beta) = \sum_{i=0}^{n} \beta_i \phi_i(x) \) (Eq. 1)
- Once a basis \( \phi \) has been selected, the hyper-parameters \( \beta \) can be selected by standard regression/optimization-based approaches.
- There can be many choices of basis functions; for simplicity, we focus on terms that are low-degree polynomials in either the parameter, \( x \), or the inverse of the parameter, \( 1/x \):
\[
\prod_{i=1}^{n} (x_i)^{p_i} : p_i \in \{-1,0,1\}, \ i = 1,...,n
\] (Eq. 2)
- One of our goals in building a model: Simplicity - Incorporate only a handful of basis terms, \( n \), from the set (Eq. 2).
- Given an initially empty set, we follow a greedy procedure (forward model selection approach) of adding to this set (P) the terms that most reduces the prediction error. Formally, \( p \) being a term, this means we determine those \( p \) that solves
\[
\min_{p \cup \{1,...,n\}} \sum_{y} (m(x; \beta) - y)^2
\] (Eq. 3)
- We do this until we reach a desired limit on the number of terms to include.

Results

Tuning Read Performance
- A strategy for collecting read traces and analyze for patterns
- Based on the observed patterns, create partial replicas of data in a layout that similar future read patterns would benefit
- Transparently redirect future reads to the replicas

Results

Performance improvement
- Impact of Aggregators to Stripe Count ratio

Motivation

Parallel I/O
- An essential component of modern HPC
- Scientific simulations periodically write their state as checkpoint datasets
- Input and output datasets have become larger and larger
- Optimal I/O performance is critical for utilizing the power of HPC machines

Complex Parallel I/O Software Stack
- Different optimization parameters at each layer of the I/O software stack
- Complex inter-dependencies between these layers
- Highly dependent on the application, HPC platform, and problem size/concurrency
- Application developers need good I/O performance without becoming experts on the I/O

Tuning Writing Data with Model-based training
- Three Phases
  - Training phase: Using a training set, develop an empirical write performance model
  - Pruning phase: Using the model predicted write times select the top k configurations that have the best performance. Store the tuned parameters of the best performing configuration
  - Refitting phase (optional): Refit the model to obtain the current characteristics of the I/O system

Performance Modeling and Tuning Writes
Nonlinear regression model
- We consider smooth, nonlinear models, which can be written as linear combinations of \( n \) nonlinear basis functions \( \phi \).
- \( m(x;\beta) = \sum_{i=0}^{n} \beta_i \phi_i(x) \) (Eq. 1)
- Once a basis \( \phi \) has been selected, the hyper-parameters \( \beta \) can be selected by standard regression/optimization-based approaches.
- There can be many choices of basis functions; for simplicity, we focus on terms that are low-degree polynomials in either the parameter, \( x \), or the inverse of the parameter, \( 1/x \):
\[
\prod_{i=1}^{n} (x_i)^{p_i} : p_i \in \{-1,0,1\}, \ i = 1,...,n
\] (Eq. 2)
- One of our goals in building a model: Simplicity - Incorporate only a handful of basis terms, \( n \), from the set (Eq. 2).
- Given an initially empty set, we follow a greedy procedure (forward model selection approach) of adding to this set (P) the terms that most reduces the prediction error. Formally, \( p \) being a term, this means we determine those \( p \) that solves
\[
\min_{p \cup \{1,...,n\}} \sum_{y} (m(x; \beta) - y)^2
\] (Eq. 3)
- We do this until we reach a desired limit on the number of terms to include.

Results

Tuning Read Performance
- A strategy for collecting read traces and analyze for patterns
- Based on the observed patterns, create partial replicas of data in a layout that similar future read patterns would benefit
- Transparently redirect future reads to the replicas

Results

Performance improvement
- Impact of Aggregators to Stripe Count ratio

Motivation

Parallel I/O
- An essential component of modern HPC
- Scientific simulations periodically write their state as checkpoint datasets
- Input and output datasets have become larger and larger
- Optimal I/O performance is critical for utilizing the power of HPC machines

Complex Parallel I/O Software Stack
- Different optimization parameters at each layer of the I/O software stack
- Complex inter-dependencies between these layers
- Highly dependent on the application, HPC platform, and problem size/concurrency
- Application developers need good I/O performance without becoming experts on the I/O

Tuning Writing Data with Model-based training
- Three Phases
  - Training phase: Using a training set, develop an empirical write performance model
  - Pruning phase: Using the model predicted write times select the top k configurations that have the best performance. Store the tuned parameters of the best performing configuration
  - Refitting phase (optional): Refit the model to obtain the current characteristics of the I/O system

Performance Modeling and Tuning Writes
Nonlinear regression model
- We consider smooth, nonlinear models, which can be written as linear combinations of \( n \) nonlinear basis functions \( \phi \).
- \( m(x;\beta) = \sum_{i=0}^{n} \beta_i \phi_i(x) \) (Eq. 1)
- Once a basis \( \phi \) has been selected, the hyper-parameters \( \beta \) can be selected by standard regression/optimization-based approaches.
- There can be many choices of basis functions; for simplicity, we focus on terms that are low-degree polynomials in either the parameter, \( x \), or the inverse of the parameter, \( 1/x \):
\[
\prod_{i=1}^{n} (x_i)^{p_i} : p_i \in \{-1,0,1\}, \ i = 1,...,n
\] (Eq. 2)
- One of our goals in building a model: Simplicity - Incorporate only a handful of basis terms, \( n \), from the set (Eq. 2).
- Given an initially empty set, we follow a greedy procedure (forward model selection approach) of adding to this set (P) the terms that most reduces the prediction error. Formally, \( p \) being a term, this means we determine those \( p \) that solves
\[
\min_{p \cup \{1,...,n\}} \sum_{y} (m(x; \beta) - y)^2
\] (Eq. 3)
- We do this until we reach a desired limit on the number of terms to include.

Results

Tuning Read Performance
- A strategy for collecting read traces and analyze for patterns
- Based on the observed patterns, create partial replicas of data in a layout that similar future read patterns would benefit
- Transparently redirect future reads to the replicas

Results

Performance improvement
- Impact of Aggregators to Stripe Count ratio