Adaptive auto-tuning in HPX using APEX

Mohammad Alaul Haque Monil[1], Bibek Wagle[2], Kevin Huck[3], Hartmut Kaiser[2]

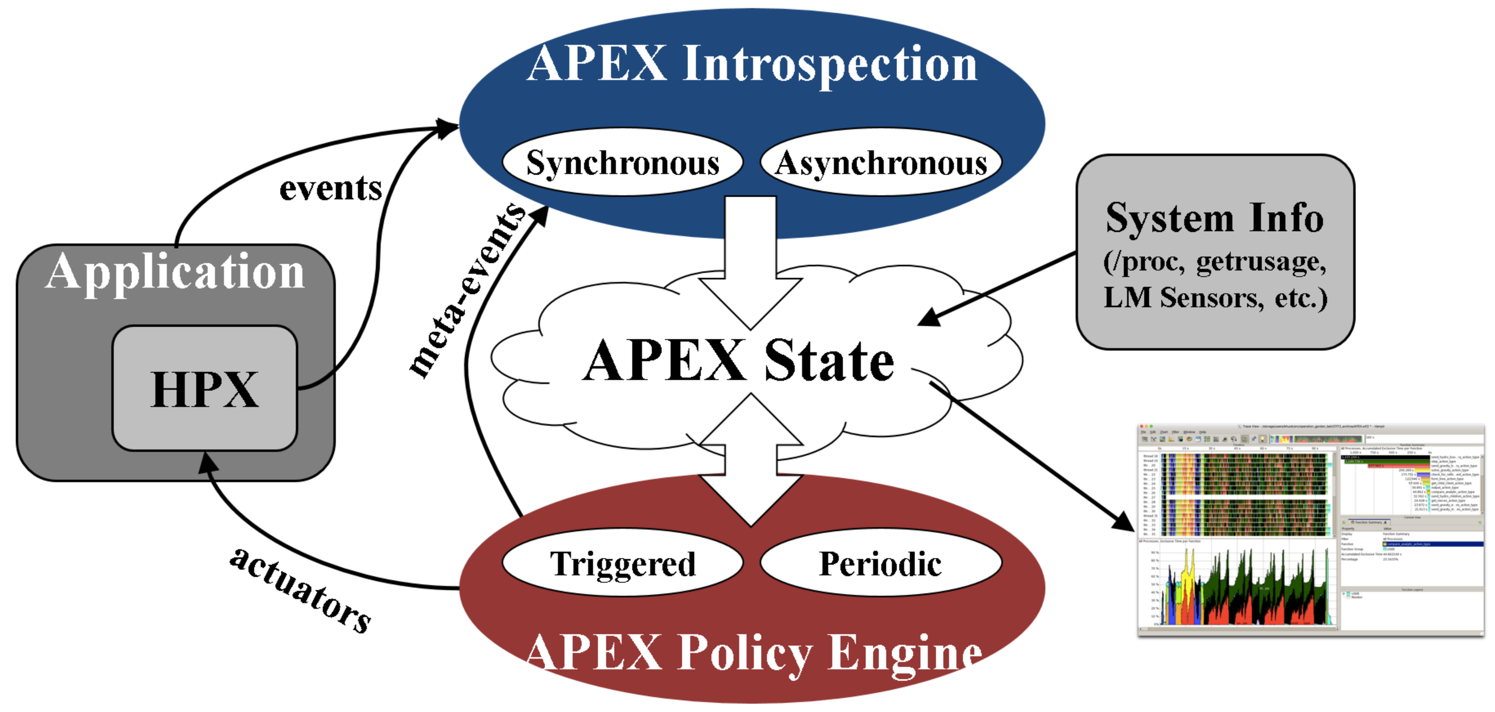
[1]Department of Computer and Information Science , [3]Performance Research Laboratory, University of Oregon, Eugene, Oregon.
[2]School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, Louisiana.

Abstract

Finding an optimal value for a parameter that impacts application performance is a hard problem and often requires repetitive execution and hence incurs wastage of resources. In this research, we provide a preliminary study which demonstrates parameter value searching at runtime for better performance. We use APEX performance measurement library to implement adaptive auto-tuning policy to tune parcel coalescing parameters based on a sampled counter of HPX runtime system.

APEX

- A performance measurement library for distributed, asynchronous tasking models/runtimes. i.e. HPX, but there are others. [1]
- Lightweight measurement (tasks <1ms) and High concurrency.
- Distinction between OS and runtime (HPX) thread context
- Lack of a traditional call stack, task dependency chain instead
- Runtime controlled task switching
- Infrastructure for dynamic feedback and control of both the runtime and the application



APEX Introspection and Event Listener:

- APEX collects data through "inspectors": Synchronous uses an event API and event "listeners". Asynchronous do not rely on events, but occur periodically based on Sampled values (counters from HPX).
- It exploits access to performance data from lower stack components: "Health" data through other interfaces (/proc/stat, /proc/cpuinfo, etc.)
- Profiling listener: Capture parent task relationship, Start, Stop event, etc.
- TAU and OTF2 Listener (postmortem analysis): Synchronously passes all measurement events to TAU and libotf2 to build an offline profile/trace analysis.
- Concurrency listener (postmortem analysis): Start event: push timer ID on stack and Stop event: pop timer ID off stack.

APEX Policy Engine :

- Policies are rules that decide on outcomes based on observed state.
- Triggered policies are invoked by introspection API events.
- Periodic policies are run periodically on asynchronous thread.
- All Policies are registered with the Policy Engine with a callback function. Callback functions define the policy rules. "If x < y then…" – any arbitrary logic.
- Enables runtime adaptation using introspection data through feedback and control mechanism and engages actuators across stack layers.
- Active Harmony is integrated for adaptive auto-tuning.

Parcel Coalescing in HPX

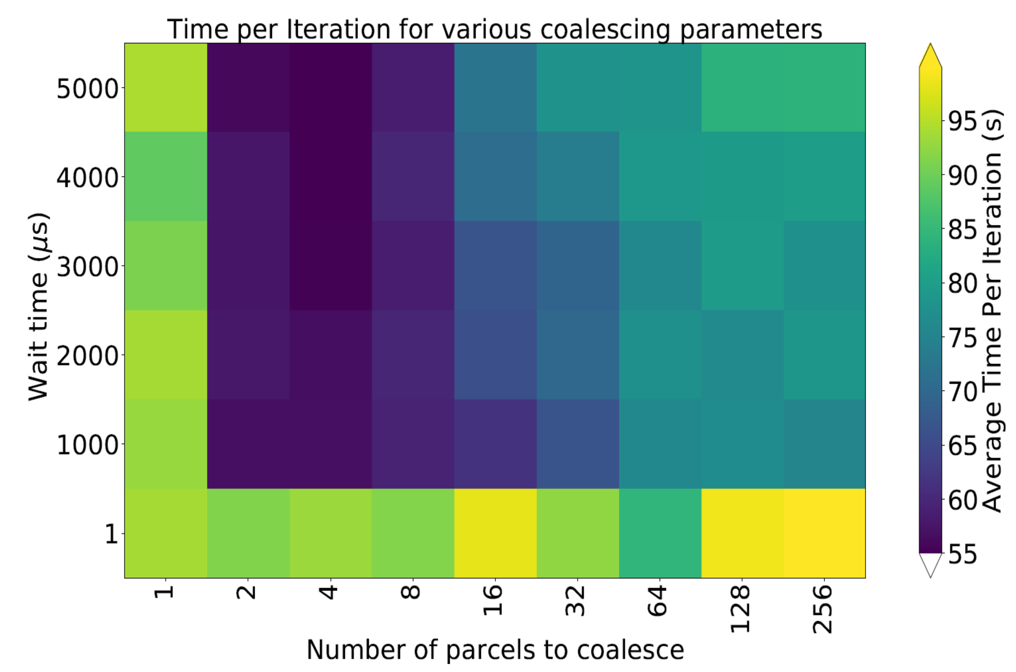
HPX is a C++ runtime system based on the ParalleX model. The HPX threading system employs lightweight tasks, known as HPX threads, that are scheduled on top of operating system threads. In a distributed environment, a locality in HPX is an abstraction for a physical node. The Active Global Address Space (AGAS) system in HPX provides a mechanism for addressing any HPX object globally.

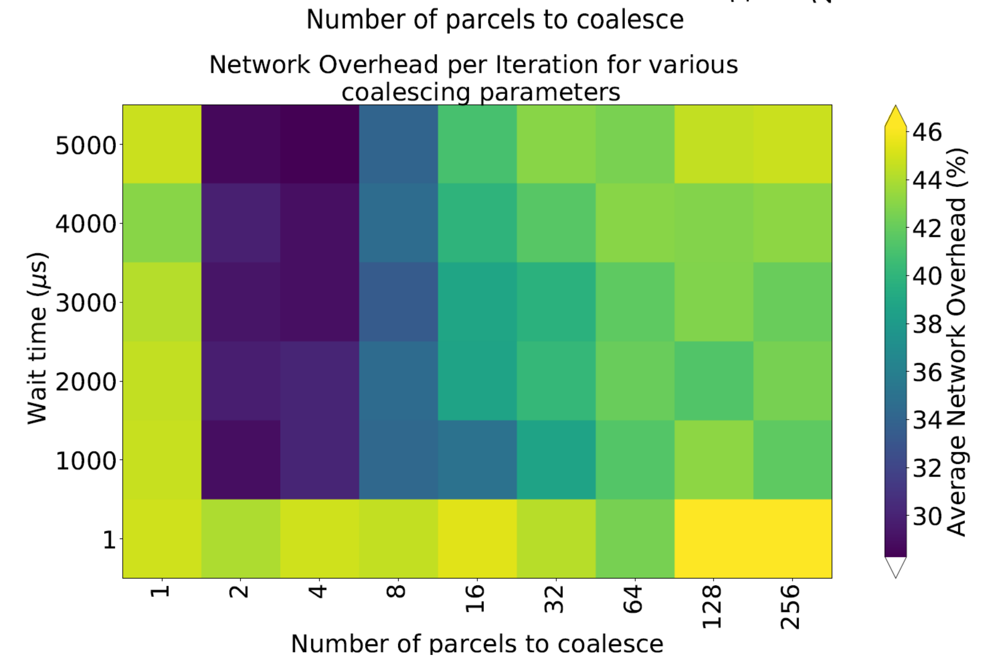
- Because of lightweight tasks in HPX, it produces fine-grained communication.
- Parcel coalescing technique is used in HPX to reduce overhead.
- Two parameters: number of Parcel and coalescing interval drives the algorithm.
- Performance depends on the value of these parameters.
- In [2], a positive correlation between task overhead and overall execution time is found.
- Network overhead is the ratio of background work (network related overhead) and total task duration and this is an HPX counter.
- Parquet Application, a complex physics simulation is tested for different wait time and number of messages to coalesce.
- The first graph represents heat map of execution time and the second one represents average network overhead.
- Two graphs show similar heat maps which show the correlation between execution time and network overhead.
- The application was run many times to find out this result.

Algorithm 1 Parcel Coalescing

procedure COALESCING MESSAGE HANDLER
  $nparcels \leftarrow$ *number of parcels to coalesce in a message*
  $interval \leftarrow$ *wait time in microseconds*
  $s \leftarrow$ *state of arriving parcel*
  $tslp \leftarrow$ *time since last parcel*
  if $tslp > interval$ **then**
    send parcel
  switch s **do**
    case $First$:
      Start *Flush timer*
      Queue Parcel
    case $!First || Last$:
      Queue Parcel
    case $Last(QueueFull)$:
      Stop *Flush timer*
      Flush queued parcels

$$n_{oh} = \frac{\sum t_{background-work}}{\sum t_{func}}$$

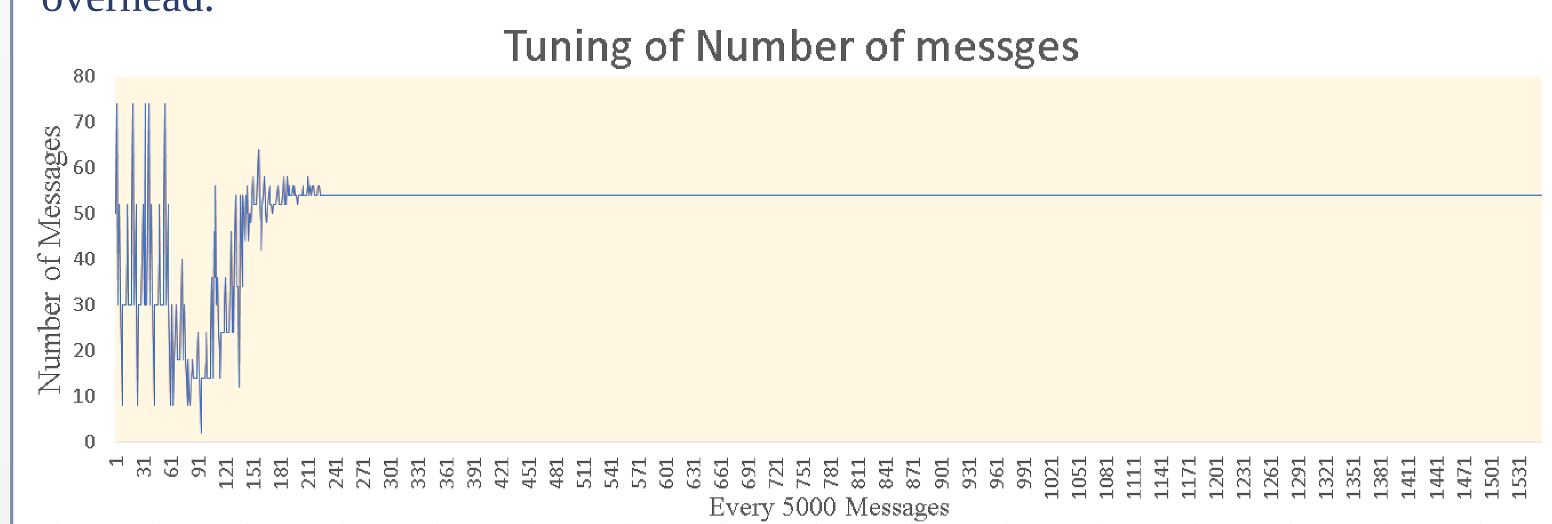


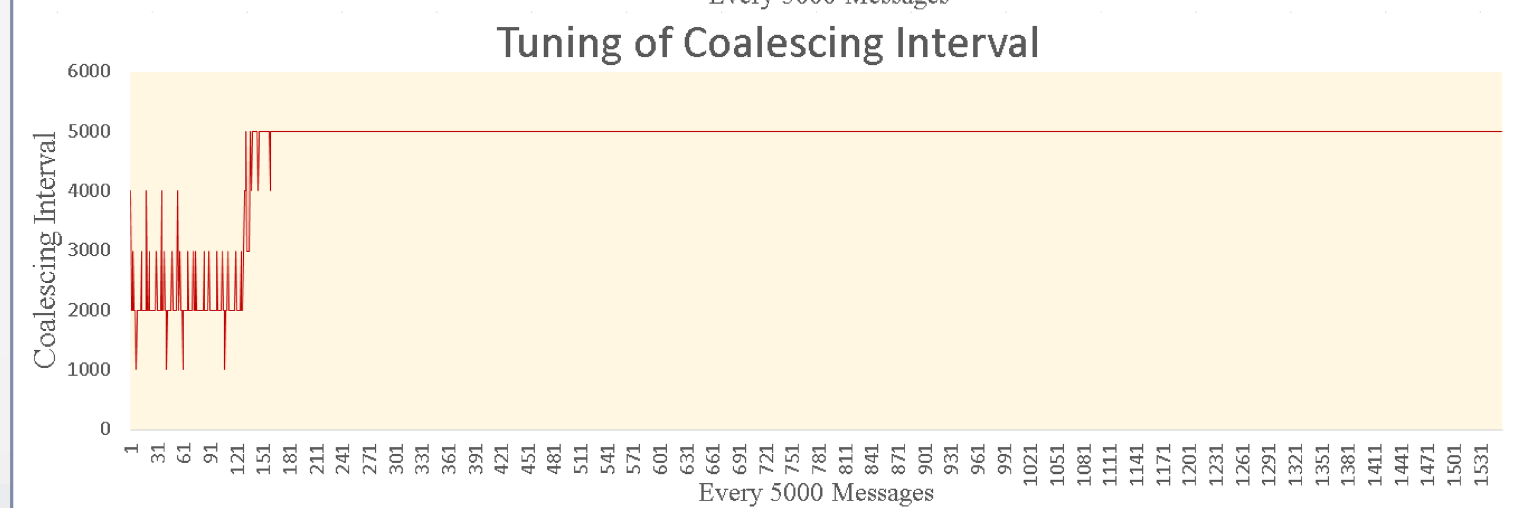


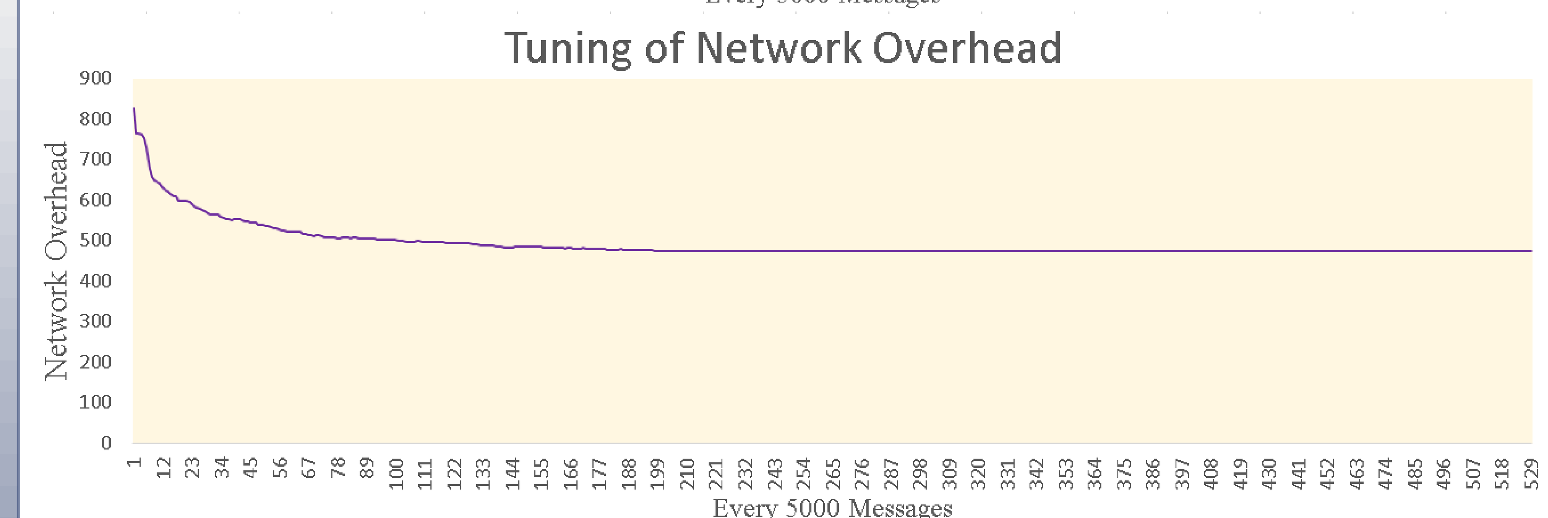
- This finding brings the opportunity for adaptive APEX policy where APEX policy will find the suitable values for wait time and number message to coalesce during the application runtime.

Adaptive Parcel Coalescing Policy in APEX

- To avoid repetitive execution to search for the best parameter values we defined a Parcel Coalescing Policy.
- We have the option to trigger the policy periodically or based on an event, for example: every 5000 messages.
- The application starts with a default/random/user_provided starting values for the interval and the number of messages to coalesce.
- The callback function for the policy is a call to Active harmony with the APEX sampled counter value of network overhead of HPX and the current value of interval and number of messages to coalesce.
- Active harmony observes the counter value to change the value of the two parameters.
- Below figures represent the impact of the policy on a toy application [2] where policy is triggered every 5000 message send events between two nodes.
- It shows that It convergence of the two parameters while reducing the network overhead.







Future Work

Apex policy shows the convergence and reduction of network overhead and provides the proof of concept of this research. We plan to test this policy for a couple of real application in large scale. Moreover, we would put more effort to find an adaptive approach to trigger this policy based on application characteristics.

Reference

1. K. A. Huck, A. Porterfield, N. Chaimov, H. Kaiser, A. D. Malony, T. Sterling, and R. Fowler. An autonomic performance environment for exascale. *Supercomputing frontiers and innovations*, *2*(3), pp.49-66, 2015
2. B. Wagle, S. Kellar, A. Serio and H. Kaiser. Methodology for Adaptive Active Message Coalescing in Task Based Runtime Systems. International Workshop on Automatic Performance Tuning. IWAPT-2018. (accepted)