Multi-threaded Object Streaming

Performance Study using CMS Conditions

Andreas Pfeiffer, CERN-CMS

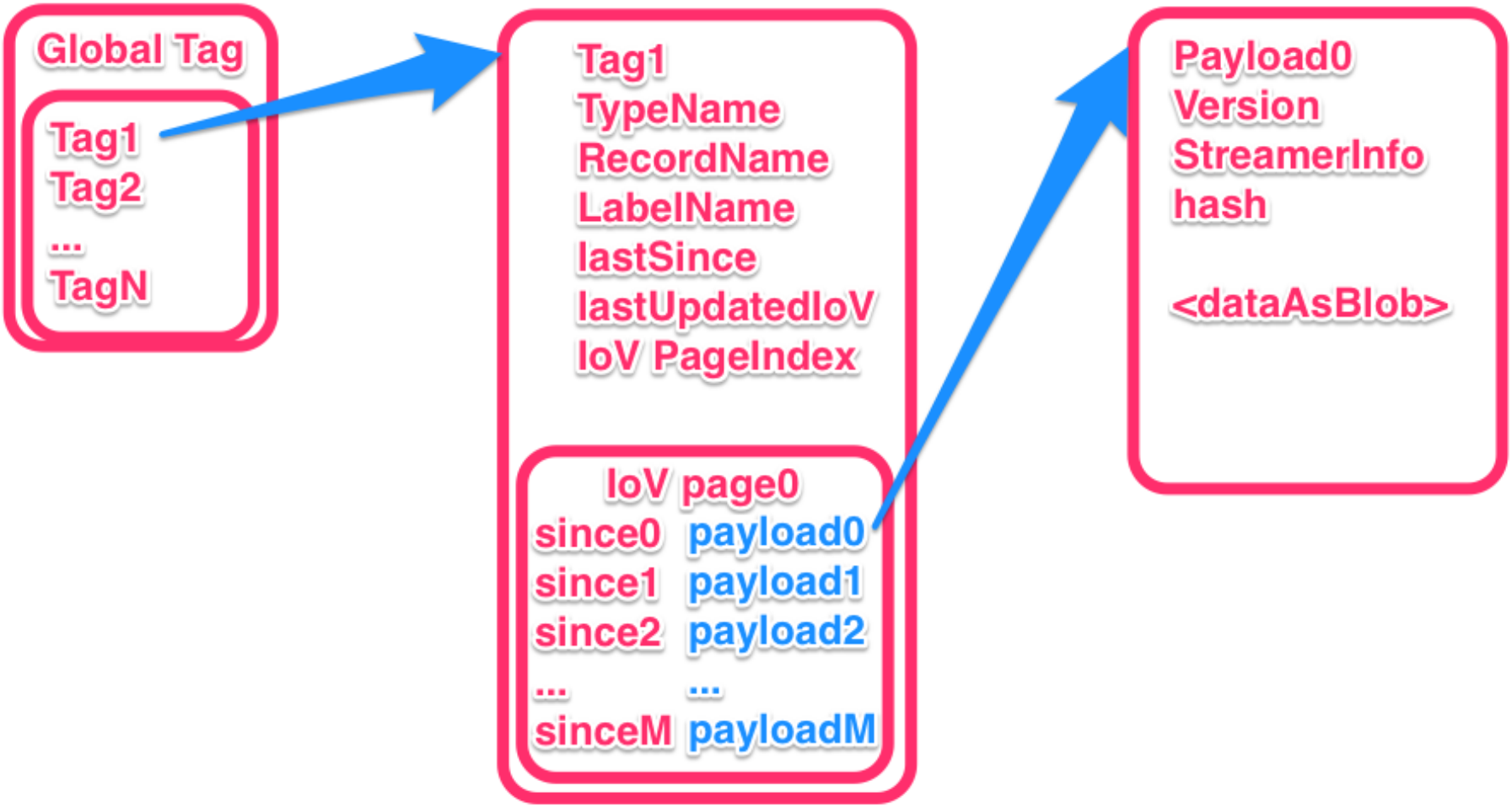
Outline

- Introduction
- Environment
- Usage
- Performance
- Summary

Context

- “Conditions” in CMS
  - non-event-related (“slow control”) information (Alignment, Calibration, Temperature, …)
  - crucial parameters for simulation/reconstruction/analysis software
  - during LS1 worked on re-design involving “lessons learned” from run 1
- CMS Framework moving to multi-threaded execution model reading/ updating conditions no longer “as needed” but at “lumi” boundaries
  - change conditions code to centrally load all conditions for a given run/lumi/timestamp
  - opportunity to use/study possible improvements of multithreaded handling of conditions

CMS Conditions - Structure



CMS Conditions

- "Global Tag"
  - coherent collection of "Tags" used for running
- "Tag"
  - identified by "Record" (and optionally "Label")
  - contains a list of "IoV"s
- "IoV" - "Interval of Validity"
  - "time" (or run, run-lumi) range (a.k.a. "since")
  - each interval has a unique "since" from which it is valid and contains a pointer to the payload object (conditions) associated with that time-interval (until the next "since")
- "Payload"
  - Conditions object stored in DB (as a "blob" in the new structure)

1 GT
≈
300 Tags

1 Tag
≈
100-10000 IoVs

Payload sizes:
100 B - O(100) MB

Multithreaded handling of CMS Conditions

- Initial study to use ***boost serialisation*** package showed promising results to store CMS Conditions using “blobs” in DB
  - easy to use (with the help of some Python and libclang)
  - good performance (comparable to root-5)
  - modern technology/tool, modular, well supported
  - potential *performance gain* using multiple threads to load and de-serialise payloads on modern multi-core boxes

Environmental Setup

- Using ***boost*** libraries for *serialization* (and *iostreams*)
  - already available in CMSSW tools
- Added "third party" product for the "Archive"
  - "portable binary archive"
    - http://epa.codeplex.com
  - ***header-only*** package, using boost
- Comparing with ROOT(5) serialisation
  - initially not using StreamerInfo

Usage - user-side

- Decided to go the “minimal intrusive” way
- For each class to be stored:
  - add/include one header file
  - add one macro in the class
  - for (non-STL container) template class X<Y> add one line in special file for each actual instance
    - 31 instances of 9 templates in 5 packages
  - for transient members: simply add a “COND_TRANSIENT” macro:
    - mutable Value_t total COND_TRANSIENT;

Usage - tool side

- Use tools available from (llvm) compiler: *libclang*
  - … and it's Python binding (***pyclang***)
    - giving access to the ***full*** **A**bstract **S**ource **T**ree (**AST**) of the code inspected
- Wrote a script (automatically/scram) to generate the (de-)serialisation code
  - small: about 500 lines of code overall
  - very flexible

Example code - user side

```
#ifndef CondFormats_PhysicsToolsObjects_Histogram_h
#define CondFormats_PhysicsToolsObjects_Histogram_h

#include "CondFormats/Serialization/interface/Serializable.h"

template<typename Value_t, typename Axis_t = Value_t>
class Histogram {
      // …
      // transient cache variables
      mutable Value_t total COND_TRANSIENT;       //CMS-THREADING
#if !defined(__CINT__) && !defined(__MAKECINT__) && !defined(__REFLEX__)
      mutable std::atomic<bool> totalValid COND_TRANSIENT;
#else
      mutable bool totalValid COND_TRANSIENT;
#endif

      COND_SERIALIZABLE;
}; // end class<> Histogram
```

Example code - generated

Histogram.h

```
template<typename Value_t, typename Axis_t = Value_t>
class Histogram {
   public:
     typedef PhysicsTools::Calibration::Range<Axis_t> Range;
     // …
   protected:
     std::vector<Axis_t>   binULimits;
     std::vector<Value_t>  binValues;
     Range                 limits;
     // …
};
```

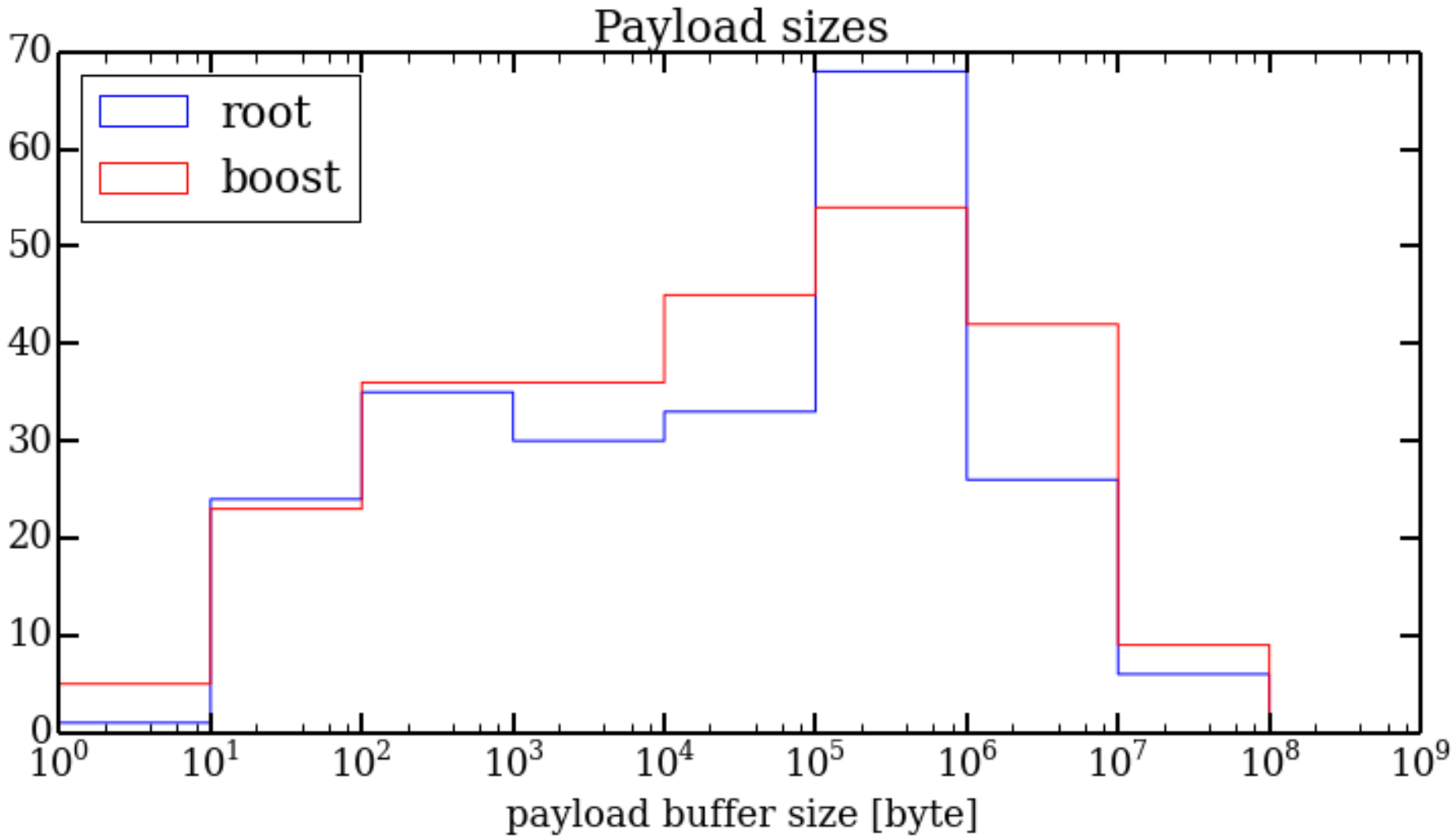
Serialization.cc (generated)

```
template <typename Value_t, typename Axis_t>
template <class Archive>
void PhysicsTools::Calibration::Histogram<Value_t, Axis_t>::serialize(
          Archive & ar, const unsigned int) {
   ar & BOOST_SERIALIZATION_NVP( binULimits );
   ar & BOOST_SERIALIZATION_NVP( binValues );
   ar & BOOST_SERIALIZATION_NVP( limits );
}
```

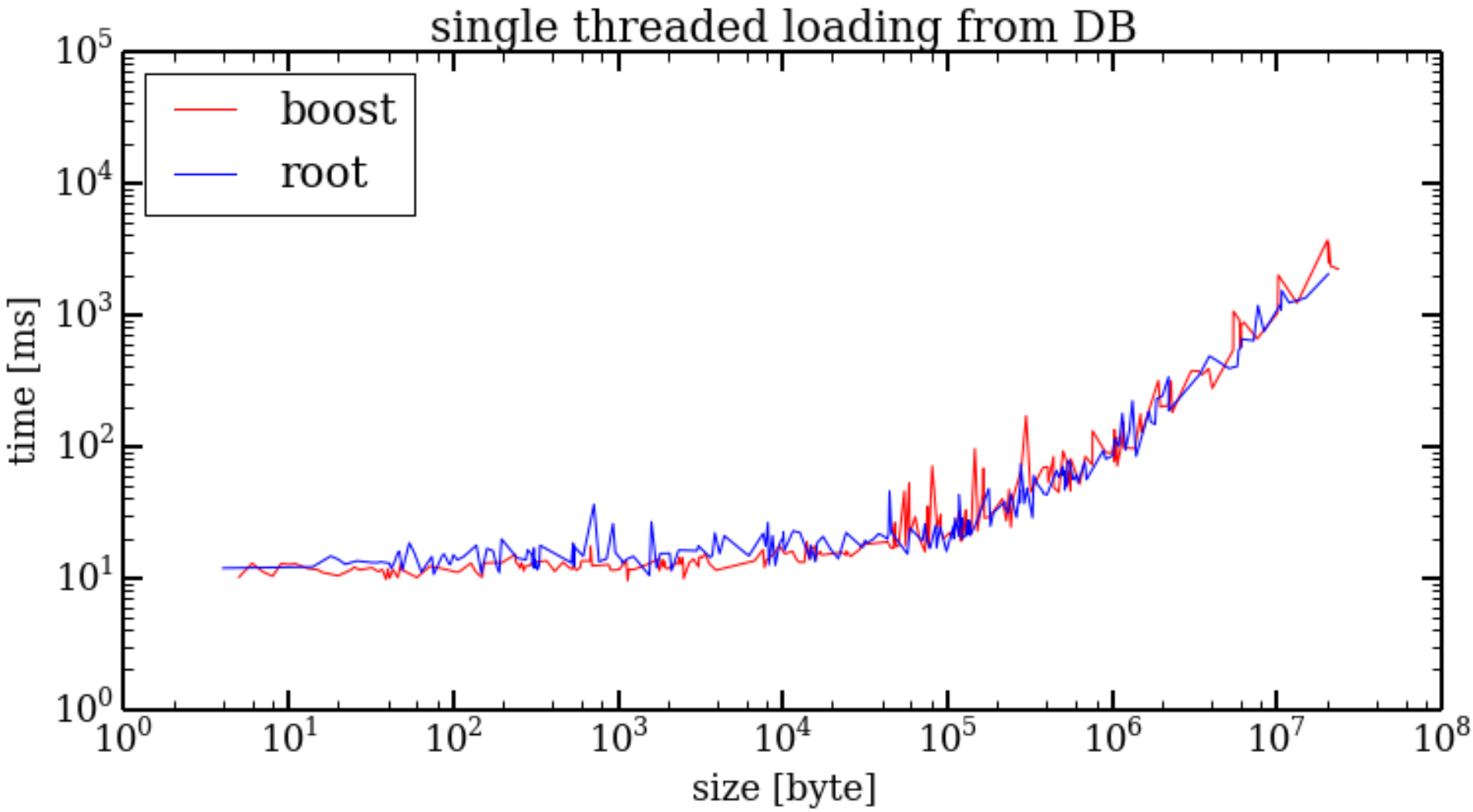
Performance study - setup

- Load all payloads for one Global Tag in dedicated program
  - START70_V1 for some run/lumi/timestamp (“random”)
    - 312 tags, overall ~300 MB of payload data (blobs)
- Procedure: Load all IOVs for all Tags of the GT, *then*
  - **fetch** all payload blobs from DB *each into it’s own buffer*, *then*
  - **deserialise** all *payload buffers* into their C++ objects.
  - ignore first run (setting up DB cache), repeat 10 times, average
- Do this for BOOST serialised conditions
  - use *parallel threads for deserialisation* use *one thread per payload (buffer)*
  - compare with ROOT(5) serialised conditions (single thread)

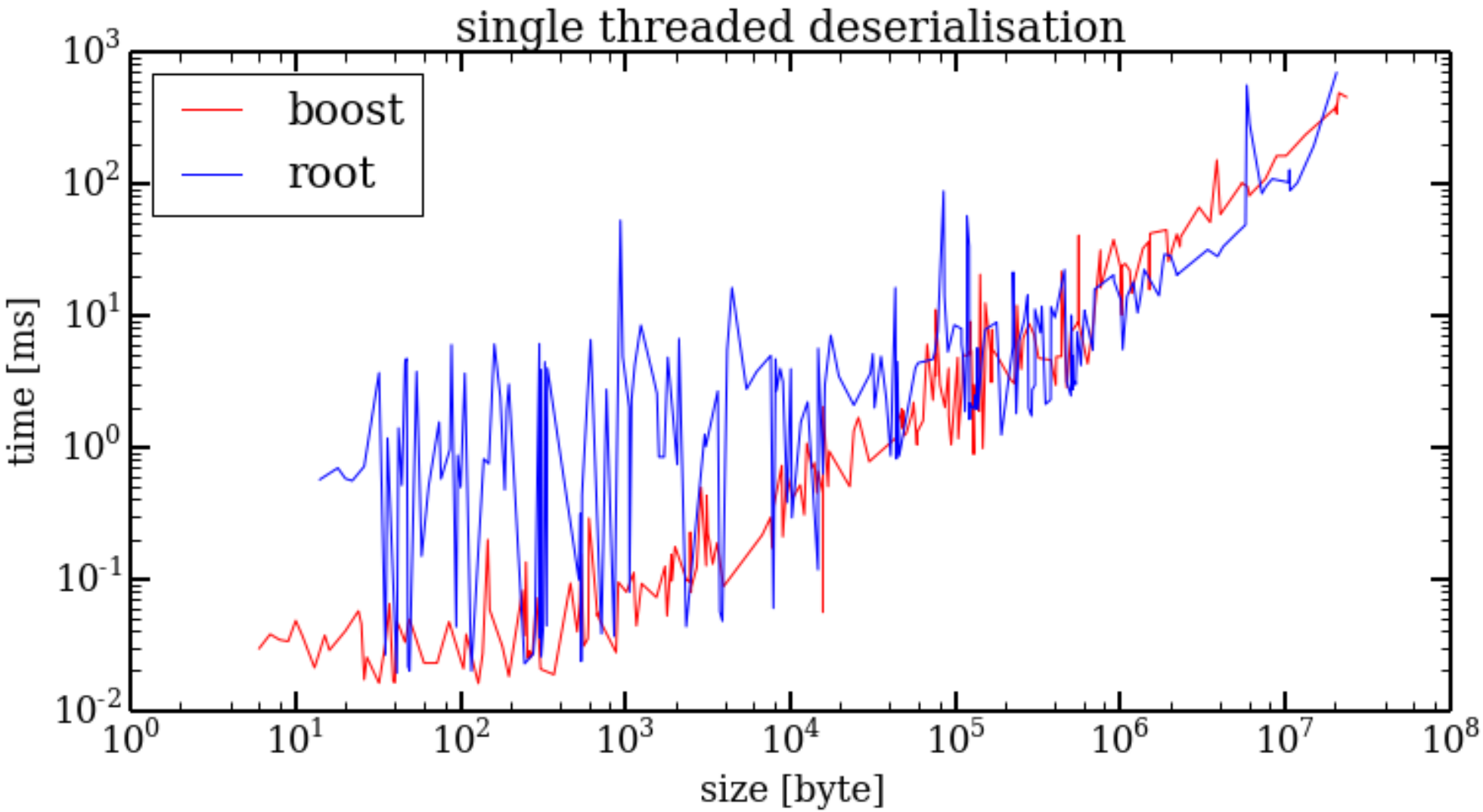
Payload (blob) sizes



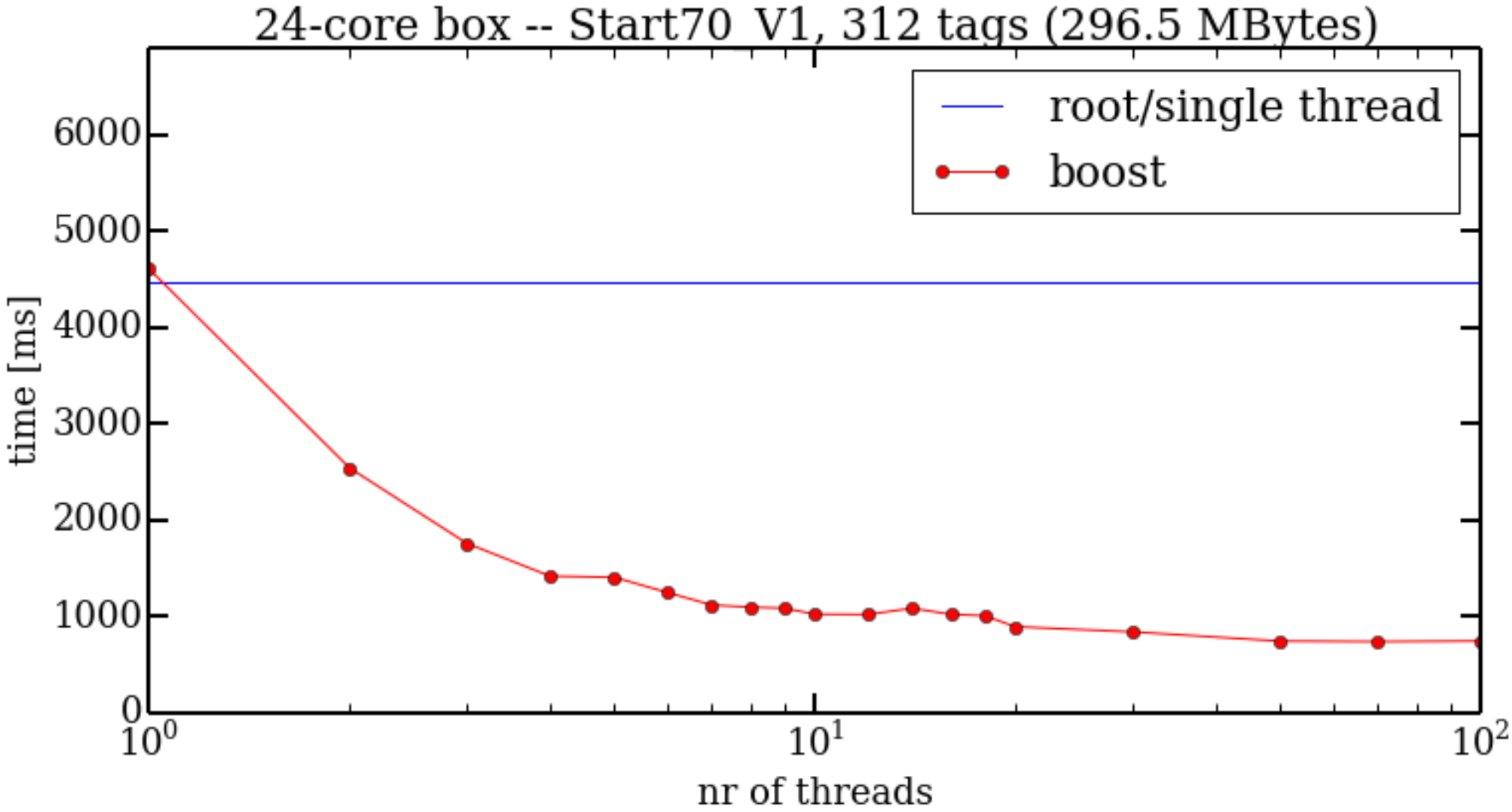
Single-treaded mode: time for fetching payloads from DB as function of payload(blob) size



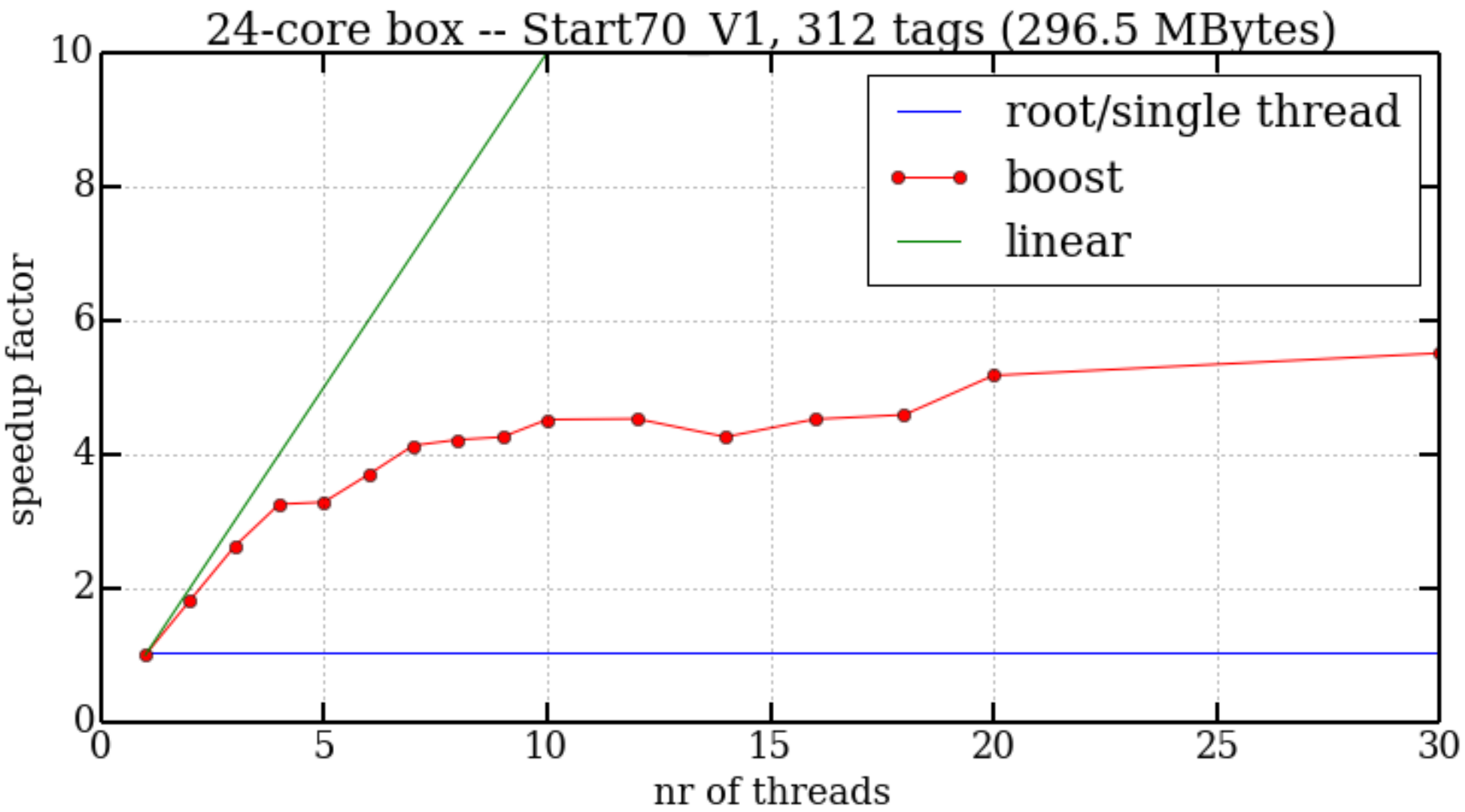
Single-treaded mode: time for deserialisation payloads as function of payload(blob) size



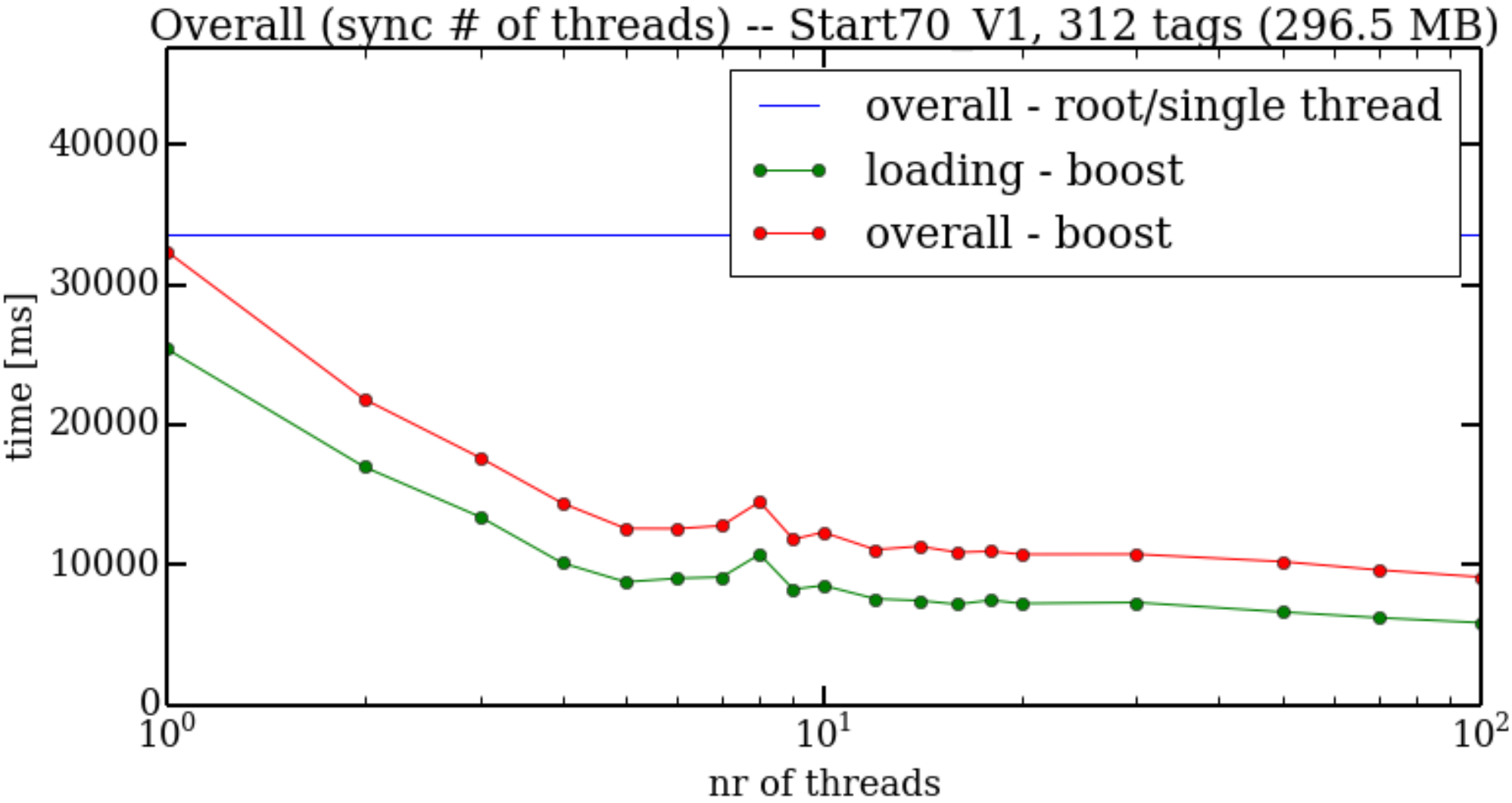
Multi-threaded deserialisation



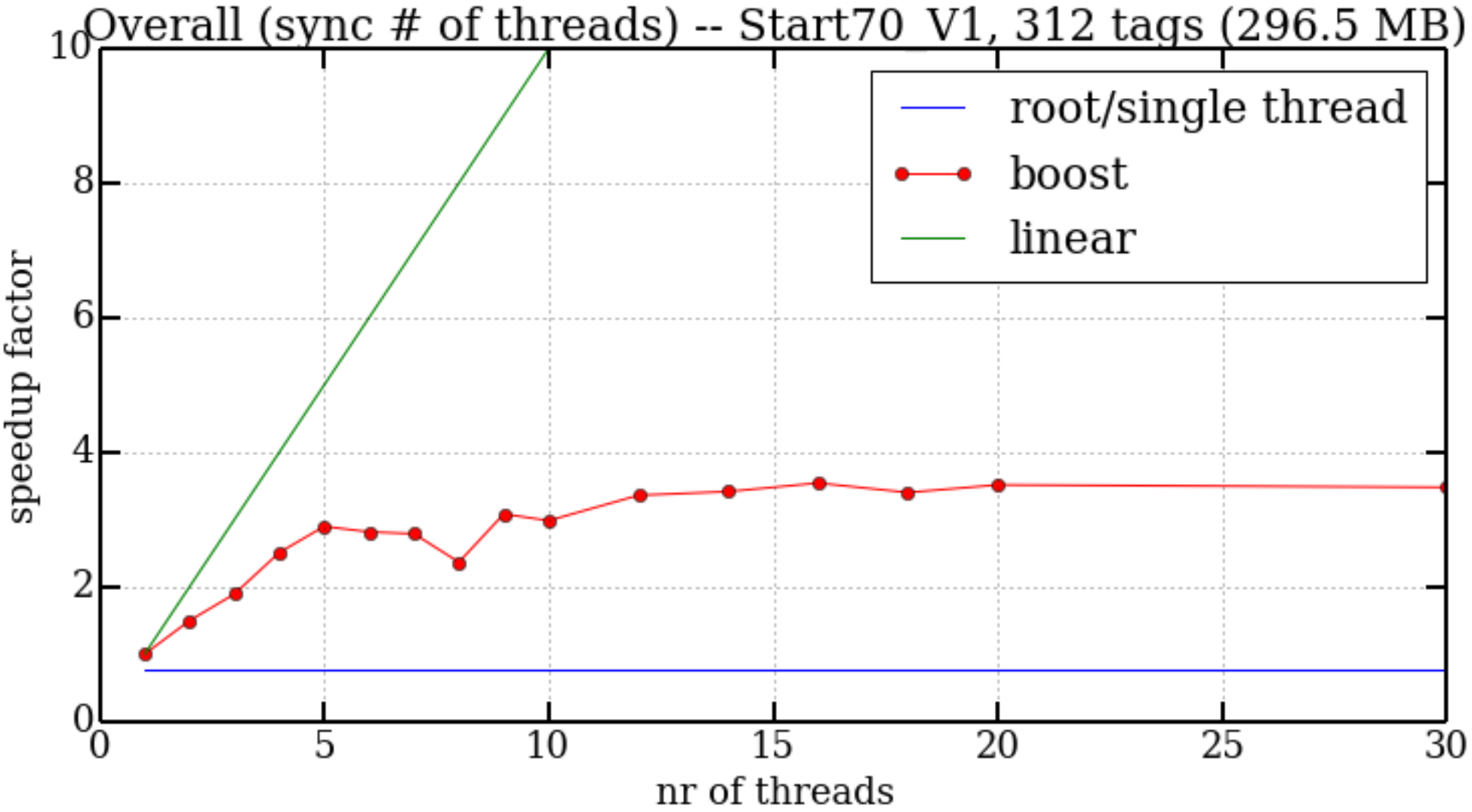
MT deserialisation - Speedup



Multi-threaded loading



MT loading - Speedup



Results

nTreads (fetch&deserial)	1	8	16	100	gain 8/1	gain 16/1	gain 100/1
loading IOVs	2723	2557	2570	2612	1.1	1.1	1.0
loading payloads	25409	10737	7173	5862	**2.4**	**3.5**	**4.3**
deserialising payloads	4233	1190	1134	648	**3.6**	**3.7**	**6.5**
overall time elapsed	32366	14485	10878	9122	2.2	3.0	3.5

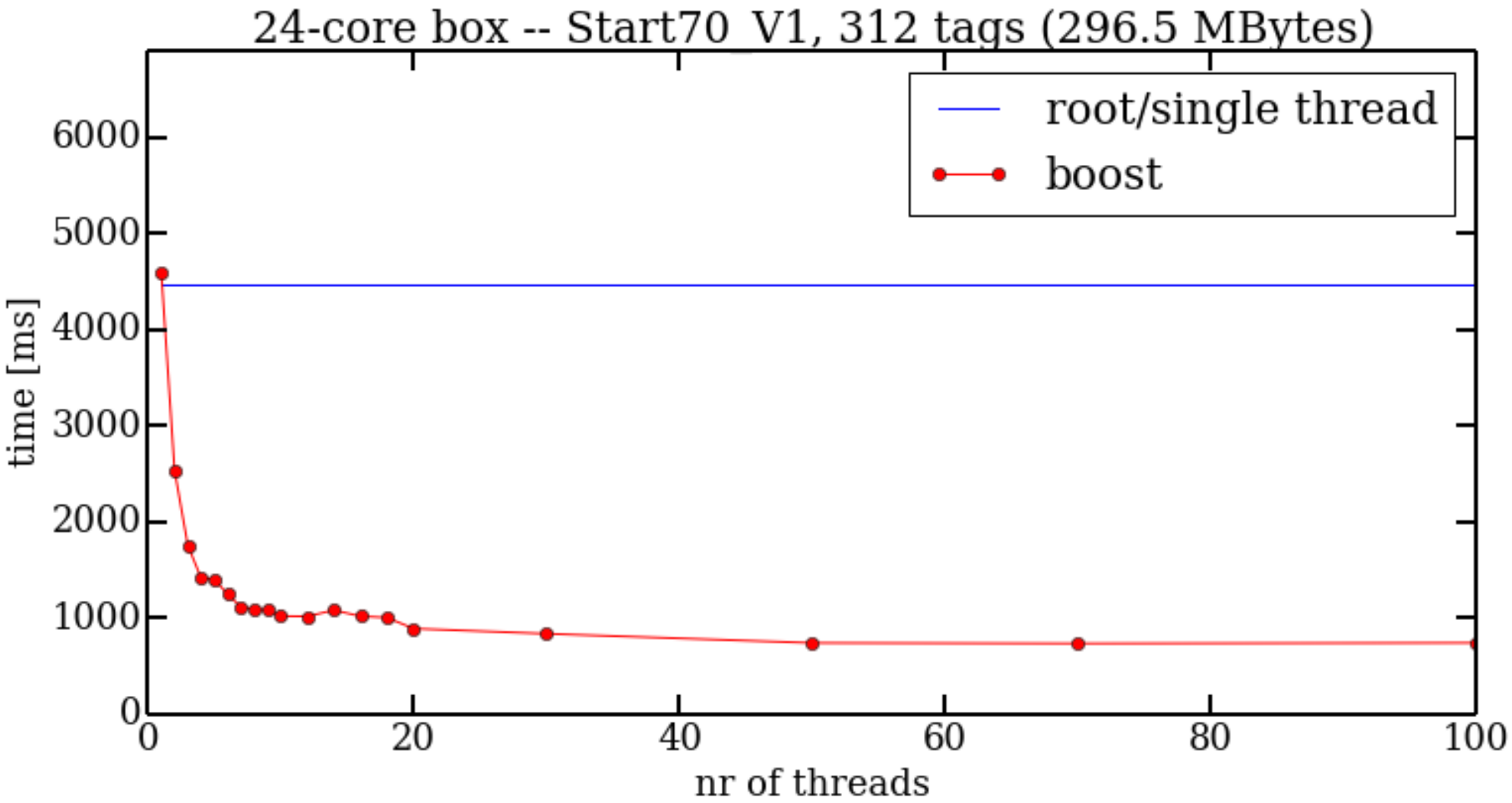
all times in msec
measured on (otherwise empty) 24-core machine

Summary

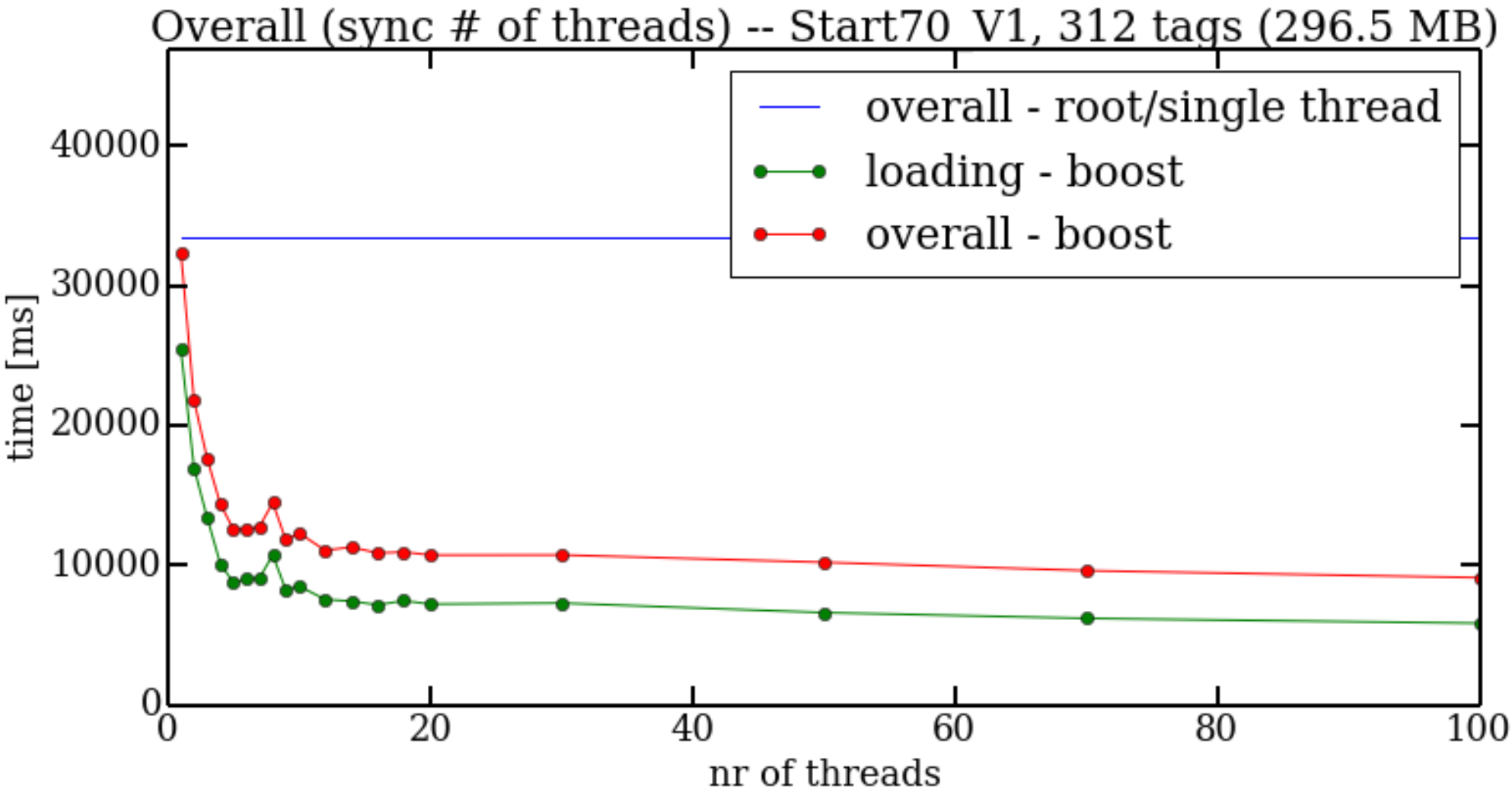
- Performance study for new CMS Conditions show promising results
  - ***multi-threaded*** running results in **speedup** of about a **factor of 3-4** for about 8-10 threads
    - both in ***loading*** payloads from DB (8s/25s) and in ***deserialisation*** (1.1s/4.2s)
- Both (de-)serialisation technologies (ROOT(5) and BOOST/ pyclang) have a similar level of “user-friendliness”
  - generated “streamer code”
- Need to measure also using Frontier (not yet MT capable)
  - also in specific online/HLT environment (only a few payloads change)

Additional slides

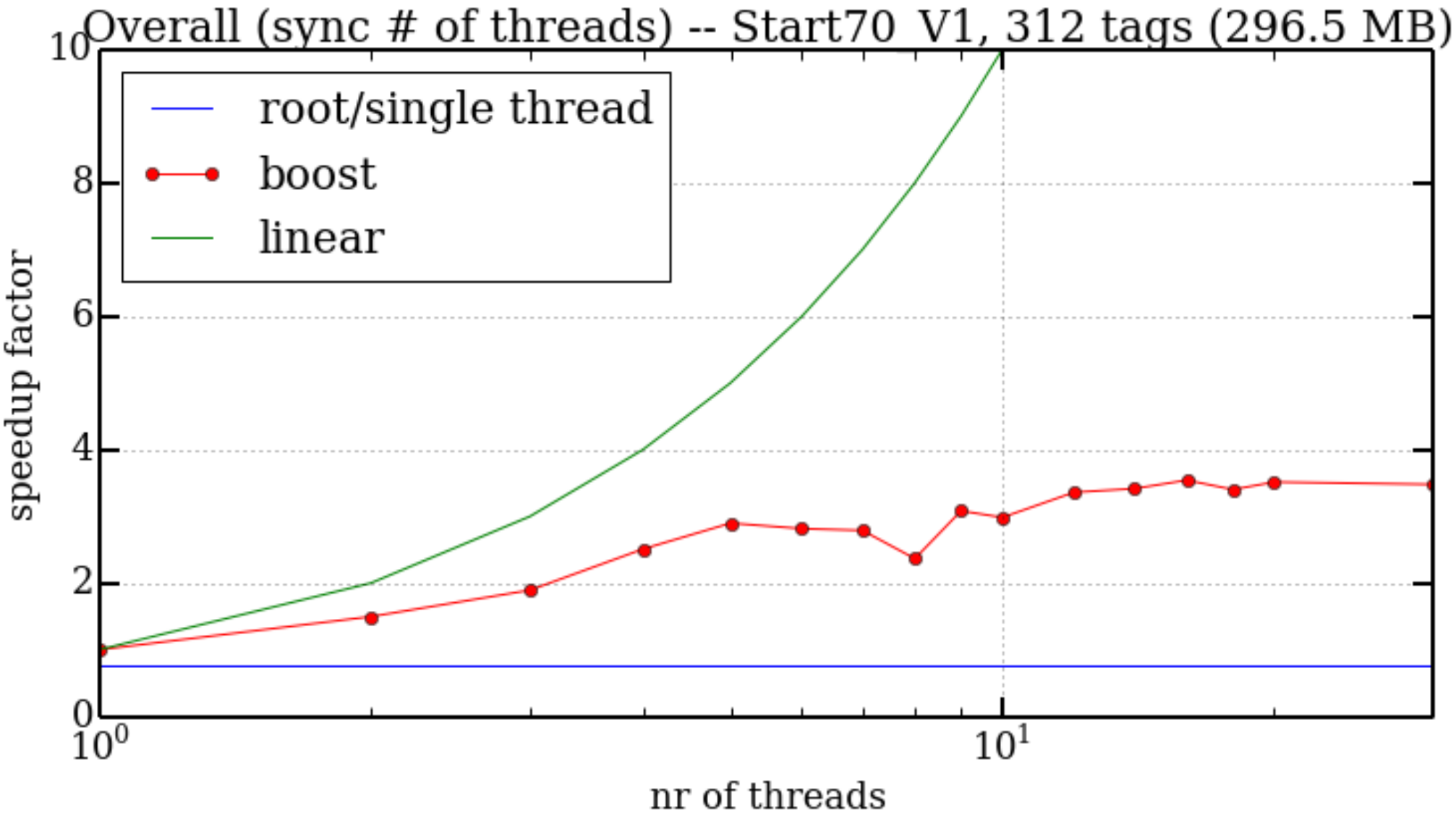
Multi-threaded deserialisation



Multi-threaded loading



MT loading - Speedup



Multi-threaded fetching - lxplus