Introducing HistFastSvc

A reasonably fast, reasonably generic alternative to HistSvc/HistNameSvc

Nicolas Morange

December 7, 2015

HistSvc + HistNameSvc

- A very nice piece of software
- Super easy to use !
- Some nice features, e.g handling of weight systematics

Not well adapted to all workflows

- David D. cutflow and optimization studies:
  - Create a small ntuple using the Reader
  - Run over it to create thousands of histograms
- Profiling showed $\sim$ 85% of CPU time in histo filling. I/O completely subdominant !
- 80% CPU time was in doing stuff with strings + looking in `unordered_maps`

$\Rightarrow$ HistSvc + HistNameSvc way too slow for such a use-case

HistFastSvc: High-level Description

Design principles

- Provide similar features as HistSvc
- Use strings only when you must
  - systematics names and sample names
- Otherwise, use integers, floats, and enums
- We can make use of a better data structure than a big flat map
- Try to cleanly separate base code from user, analysis-specific code
  - HistSvc/HistNameSvc are quite good at that too
- It's ok to **force** users to write more structured code

Intended use and caveats

- It is really tailored to fill histograms following given naming conventions
- It is less nice than plain HistSvc when filling "isolated" histos (although it could be used for that too)
- Both HistSvc and HistFastSvc can cohabit in analysis code if needed
- It really **forces** the user to write structured code

Results

- In David's workflow, total speed more than doubled
- Meaning histo "management" speed was $\sim \times 3$
- $\times 3$ confirmed with Camilla's VHRes 0 lepton code

The 1 most important thing

- The variables you want to plot should be accessible during the whole lifetime of the program, in e.g a MVATree
- The event weight as well

1. Declare your histograms

- Do it once during initialization
- Fill a vector of Histo (which are TH1/2 who know how to fill themselves)

```
EL::StatusCode AnalysisReader_BoostedVHbb::declare_fatJetAnalysisHistos () {
  if(true) {

  m_fatJetHistos.push_back({new TH1F("mVH", "mVH", 600, 0,  6000),
                            [&](){ return m_tree->mVH/ 1e3;}, m_weight});
  m_fatJetHistos.push_back({new TH1F("MET", "MET", 60, 0,  1500),
                            [&](){ return m_tree->MET / 1e3;}, m_weight});
  m_fatJetHistos.push_back({new TH1F("NBTagMatchedTrackJetLeadFatJet", "NBTagMatchedTrackJetLeadFatJet", 8, 0,  8),
                            [&](){ return m_tree->nBTagMatchTrkJetLeadFatJet;}, m_weight});
  m_fatJetHistos.push_back({new TH1F("MLeadFatJet", "MLeadFatJet", 100, 0, 500.),
                            [&](){ return m_tree->MLeadFatJet /1e3;}, m_weight});
  m_fatJetHistos.push_back({new TH1F("pTLeadFatJet", "pTLeadFatJet", 100, 0, 2500.),
                            [&](){ return m_tree->pTLeadFatJet /1e3;}, m_weight});

  if ( !m_doOnlyInputs ) {
     m_fatJetHistos.push_back({new TH1F("NSignalJets", "NSignalJets",     15, 0,  15),
                               [&](){ return m_tree->nSigJets;}, m_weight});
     m_fatJetHistos.push_back({new TH1F("NFatJets", "NFatJets",     8, 0,  8),
                               [&](){ return m_tree->nFatJets;}, m_weight});
```

2. Declare your naming conventions

- Create a small class
- Need 2 functions regName and regCode: translate parameters into a string or an int
  - Choice of parameters to pass is completely free, but of course prefer using int, float, enum, and not strings...
  - But need to be the same in the 2 functions
  - regCode: create an int unique to a given analysis category
- Need writeHistToFile and registerToELWorker
  - Used to set the layout of the output file (e.g use subdirectories...)

```
/* VHResRegions: an example of an encoding of region naming conventions
 *
 * The structure is extremely free.
 * It needs to provide
 * * int regCode() with any arguments
 * * string regName() with any arguments
 * * void writeHistToFile with specific arguments
 *
 * regCode must be some kind of hashing, i.e every (valid) set of input variables must
 * produce a unique code.
 *
 * All the rest (variables with their types) is left up to the users to tune
 *
 */
struct VHResRegions {

  enum Label { NONE=0, SR, mBBCR, mBBCRlow, mBBCRhigh, mbbPresel, mbbIncl };
  static std::map<Label, std::string> m_labelNames;

  static int regCode(int ntag, int naddTag, float pTV, float mVH, Label label=NONE);

  static std::string regName(int ntag, int naddTag, float pTV, float mVH, Label label=NONE);

  static void writeHistToFile(TH1* h, const std::string& syst, const std::string& sample,
                              const std::string& regName, TFile* f);
  static void registerToELWorker(TH1* h, const std::string& syst, const std::string& sample,
                                 const std::string& regName, EL::Worker* wk);
};
```

Break: Implementation

```
std::string VHResRegions::regName(int ntag, int naddTag, float pTV, float mVH, Label label) {
  std::string name;
  if (ntag < 0) name += "0p";
  else if (ntag == 0) name += "0";
  else if (ntag == 1) name += "1";
  else if (ntag == 2) name += "2";
  else name += "3p";
  name += "tag";

  name += "0pjet";

  // FOR BOOSTED ANALYSIS-
  // MET
  if (pTV > 150e3) { name += "_150ptv_"; }

  // mVH
  if   ( mVH < 0 ) name += "";
  else if ( mVH < 500e3 ) name += "0_500mVH_";
  else name += "500mVH_";

  // ADD BTAG
  if( naddTag >= 0 ) {
    if( naddTag == 0 ) name += "0btrkjetunmatched_";
    else  name += "1pbtrkjetunmatched_";
  }

  name += m_labelNames[label];

  return name;
}
```

```
int VHResRegions::regCode(int ntag, int naddTag, float pTV, float mVH, Label label) {
  // quite dumb, but it should work well
  if(ntag>3) { ntag = 3; }
  int tagcode = (ntag + 4)%10;
  if(naddTag>1) { naddTag = 1; }
  int njetcode = (naddTag + 4)%10;
  int pTVcode = 5;
  if (pTV <= 150e3) pTVcode = 0;
  int mVHcode = 0;
  if(mVH < 500e3) { mVHcode = 1; }
  if(mVH < 0) { mVHcode = 2; }
  int labelcode = (int)(label);

  int code = tagcode + 10 * njetcode + 100 * pTVcode + 1000 * mVHcode + 10000 * labelcode;
  return code;
}
```

```
void VHResRegions::registerToELWorker(TH1* h, const std::string& syst, const std::string& sample,
                                      const std::string& regName, EL::Worker* wk) {
  if(!wk) { /* shout something*/ return; }
  if(!h) { /* shout something*/ return; }

  // set the name
  std::string hname = sample+"_"+regName+h->GetName();
  if(syst != "Nominal") {
    hname = syst + "/" + hname;
    hname += ("_" + syst);
  }
  h->SetName(hname.c_str());
  wk->addOutput(h);
  // EL takes care of deleting histograms
}
```

3. Declare the service in your code

- Templated by the Naming class you just wrote

```
HistFastSvc<VHResRegions> m_histFastSvc; //!
HistoVec m_fatJetHistos; //!
```

4. Set sample and syst

- Until now, profit from HistNameSvc

```
EL::StatusCode AnalysisReader_BoostedVHbb::fill_0Lep ()
{
  m_tree->SetVariation(m_currentVar);
  m_histNameSvc->set_doFatJetAnalysis(true);

  // start to setup histFastSvc for this event/syst
  m_histFastSvc.ClearWeightSysts();
  m_histFastSvc.SetSample(m_histNameSvc->getFullSample());
  m_histFastSvc.SetSyst(m_currentVar, true);
```

5. Add weight systematics

- Until now, profit from HistNameSvc

```
if ( m_isMC) {
   bTagSF = computeBTagSFWeight(goodTrackJets);
   // NM add variations to HistFastSvc
   for(auto& p : m_weightSysts) {
     m_histFastSvc.AddWeightSyst(p.name, p.factor);
     //std::cout << "Adding weight syst " << p.name
   }
   m_weight *= bTagSF;
}
```

6. Fill histograms

- Just call FillAccordingTo with the vector of histos, and the parameters of your regCode/regName

```
m_histFastSvc.FillAccordingTo(m_fatJetHistos, tagcatExcl, nTrackJetUnmatched, metVec.Pt(),
                              -1, mbbRegionVH);
```

7. Write to the output

- Same as current HistSvc

```
std::cout << "about to write histos" << std::endl;
m_histFastSvc.Write(wk());
std::cout << "done writing histos" << std::endl;
```

And that's it !

Histo

- Can accept any TH1 or TH2
- Rely on pointers to variables, or functions (typically lambdas) to fill themselves

```
class Histo
{
  private:
    std::unique_ptr<TH1> m_h;
    std::unique_ptr<TH2> m_h2;
    double& m_w;
    double* const m_x;
    double* const m_y;
    std::function<double()> m_getx;
    std::function<double()> m_gety;

  public:
    // empty
    Histo() noexcept = default;
    // mpve
    Histo(Histo&&) = default;
    // copy
    Histo(const Histo& other) noexcept;

    // 1-D histos
    Histo(TH1* h, double* x, double& w) noexcept;
    Histo(TH1* h, std::function<double()> x, double& w) noexcept;

    // 2-D histos
    Histo(TH2* h, double* x, double* y, double& w) noexcept;
    Histo(TH2* h, double* x, std::function<double()> y, double& w) noexcept;
    Histo(TH2* h, std::function<double()> x, double* y, double& w) noexcept;
    Histo(TH2* h, std::function<double()> x, std::function<double()> y, double& w) noexcept;

    ~Histo();

    void Fill(double additional_w = 1.0);
    TH1* GetHistCopy();

};

using HistoVec = std::vector<Histo>;
```

HistFastSvc interface

- Keep it dead simple

```
public:
  HistFastSvc() = default;
  ~HistFastSvc() = default; // to be revisited some day by taking care of deallocating histos properly


  /// Setting stuff

  /// Nominal sample and syst
  inline void SetSyst(const std::string& sysName, bool fillWeightSysts=false) {
    if(sysName != m_syst) { m_syst = sysName; m_validCache = false;}
    m_fillWeightSysts = fillWeightSysts;
  }

  inline void SetSample(const std::string& sampleName){
    if(sampleName != m_sample) { m_sample = sampleName; m_validCache = false;}
  }

  /// Weight systs
  inline void AddWeightSyst(const std::string& sysName, double additional_w = 1.0) {
    m_weightSysts.push_back( std::make_pair(sysName, additional_w));
    m_validWeightSystsCache = false;
  }

  inline void ClearWeightSysts() {
    m_weightSysts.clear();
    m_validWeightSystsCache = false;
  }

  /// It's a kind of magic ©Queen
  template<class ...ArgTypes>
  void FillAccordingTo(const HistoVec& tplte, ArgTypes... args);

  /// Write all histos to file. How histos are renamed, or placed into subdirectories, is
  /// delegated to RegNaming
  void Write(TFile* file);

  /// Register all histos to EventLoop. How histos are renamed, or placed into subdirectories, is
  /// delegated to RegNaming
  void Write(EL::Worker* wk);
```