A simplified configuration for common algorithms for ATLAS analysis

Matthew Maroun (On behalf of the ATLAS Software and Computing Activity)

Computing in High-Energy and Nuclear Physics 2024, AGH University of Kraków, Institute of Nuclear Physics Polish Academy of Sciences, Jagiellonian University, 21 - 25 October

Introduction

The ATLAS analysis model requires users to apply many calibrations, identifications, selections, scale factors, etc. to physics objects. Doing so requires the use of Combined Performance (CP) tools, specialized ATLAS code that must be applied in a specific order, and with a wide variety of configurations, on these objects. Making easier the user interface are the CP Algorithms, which are single common interfaces, called once per event, that house one CP tool and its configurations, creating a full sequence for each object type. To make even easier the user interface, and to optimize output file size, a user-friendly configuration using a YAML file for algorithm configuration above a physics-oriented Python configuration has been implemented.

The YAML File ("Text Configuration" - User Level)

General object selection: Specify configuration attribute by "name: 'setting'"

Indents specify config blocks; Blocks can own sub-blocks (another indent)

Common analysis blocks with simple configurations

Block Order in YAML File Does Not Matter!

Flat output ntuple contains one tree

Specify prefixes of branch names associated with objects in output ntuple

Full support for including custom CP Algorithms!

```yaml
Electrons:
  - containerName: 'AnaElectrons'
    WorkingPoint:
      - selectionName: 'loose'
        noEffSF: True
        likelihoodWP: 'LooseBLayerLH'
        isolationWP: 'Loose_VarRad'
    PtEtaSelection:
        minPt: 25000.0
        maxEta: 2.47

# Common (global) services to run.
CommonServices:
    # Turn on/off systematics
    runSystematics: False

# Specify how to handle the pileup. The number of simultaneous
# collisions in the ATLAS detector creates pileup of events, the
# characteristics of which are determined by a pileup profile.
# This weights MC samples to have the same pileup profile as
# the collision data.
PileupReweighting: {}

# Apply common event quality requirements
EventCleaning:
    # prune events containing jets with no primary vertx,
    # prune events containing loose-likelihood jets,
    # etc.
    runEventCleaning: True

Output:
    treeName: 'analysis'
    # Variables associated with containers other than MET
    #   Syntax without systematics: '<Container>_NOSYS -> <branch name>'
    #   Syntax with systematics: '<Container>_%SYS% -> <branch name>'
    vars: []
    # Variables associated with MissingET container
    metVars: []
    containers:
        'mu_': 'OutMuons'
        'el_': 'OutElectrons'
        'jet_': 'OutJets'
        'met_': 'AnaMET'
        '': 'EventInfo'

AddConfigBlocks:
    # Calls `import functionName from modulePath`
    modulePath: 'MyAnalysis.MyxAODAnalysisConfig'
    functionName: 'MyxAODAnalysisConfig'
    # Name that will be used in YAML file
    algName: 'MyxAODAnalysis'
    # Will run before 'Output' algorithm
    pos: 'Output'

MyxAODAnalysis:
    # Pass continers to use when defining variables
    muons: 'AnaMuons.medium'
    electrons: 'AnaElectrons.loose'
    jets: 'AnaJets'
```

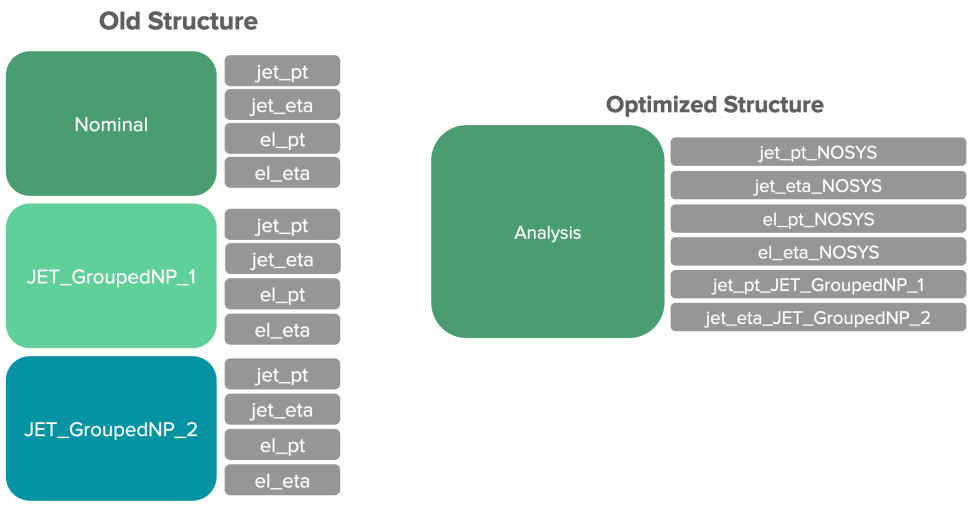
The Python Code ("Block Configuration")

```python
configSeq += config.makeConfig ('Electrons',
    containerName='AnaElectrons' )
configSeq.setOptionValue ('.forceFullSimConfig', forceEGammaFullSimConfig)
if not forCompare :
    configSeq.setOptionValue ('.recalibratePhyslite', False)
configSeq += config.makeConfig ('Electrons.WorkingPoint',
    containerName='AnaElectrons',
    selectionName='loose')
configSeq.setOptionValue ('.forceFullSimConfig', forceEGammaFullSimConfig)
if forCompare :
    configSeq.setOptionValue ('.noEffSF', True)
else:
    configSeq.setOptionValue ('.noEffSF', geometry is LHCPeriod.Run2)
if likelihood:
    configSeq.setOptionValue ('.identificationWP', 'LooseBLayerLH')
else:
    configSeq.setOptionValue ('.identificationWP', 'LooseDNN')
configSeq.setOptionValue ('.isolationWP', 'Loose_VarRad')
configSeq.setOptionValue ('.writeTrackD0Z0', True)

configSeq += config.makeConfig ('Electrons.PtEtaSelection',
    containerName='AnaElectrons')
configSeq.setOptionValue ('.selectionDecoration', 'selectPtEta')
configSeq.setOptionValue ('.minPt', electronMinPt)
configSeq.setOptionValue ('.maxEta', electronMaxEta)
```

- Information from YAML file gets parsed and set in Python config
- Each block is added to a config sequence and compiled using a config accumulator
- Attributes are set here; all unspecified attributes are set to their default values that define the config block
- The config block calls the CP Algorithm with these attributes and the corresponding CP Tool

Optimized Flat NTuple Output



- CP Algorithms greatly reduce ntuple size through its optimized output structure (by 1-2 orders of magnitude)
- Helped made possible by running systematics only on directly affected tools/objects