OSIsoft.

vCampus Live! 2012

PI AFSDK Scalability and Performance

Presented by **Ray Hall**

Demonstrate best practices for PI Server 2012

AFSDK

OSIsoft.AF.PI
OSIsoft.AF.Data

AF Server

PI Data Archive

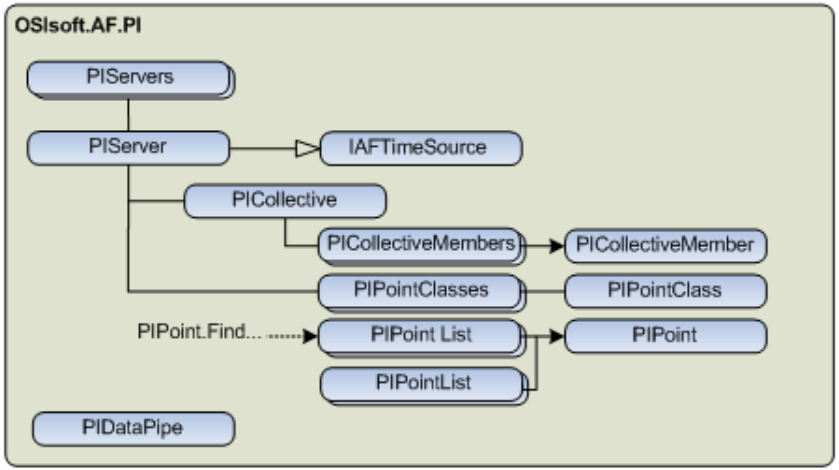
Performance
Scalability
Reliability
Manageability

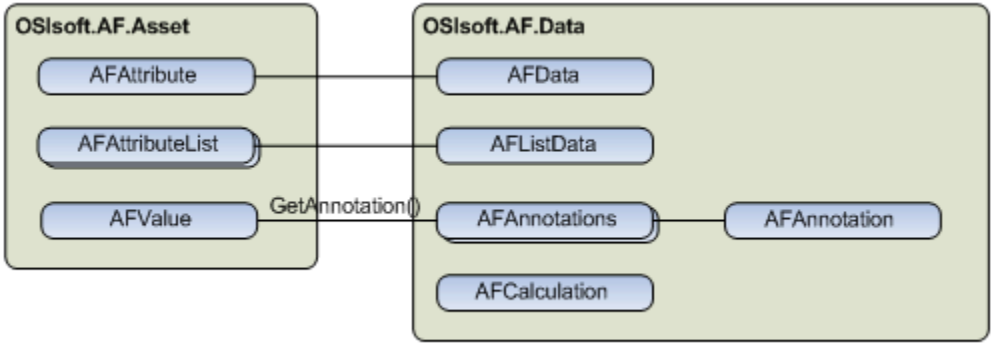
2012 vs. 2010: The Final Sheet

	2010	2012	Delta
Max Point Count	2-3M	20M+	5-10x
Startup Time	>10 min/Mpts	<30 sec/Mpts	20x
Point Creation	<100 pt/sec	500-2K pt/sec	5-200x
Tag Searching	Variable, Non-Linear	Constant or Linear	N/A
Max Update Signups	<200K	10M+	50x
Update Signup Rate	<2K/sec	>100K/sec	50x
Data Out (Archive)	<1M ev/sec	>10M ev/sec	10-20x
Data In (Snapshot)	<200K ev/sec	>1M ev/sec	5-10x
Data In (Archive)	<100K ev/sec	>500K ev/sec	5-10x
Archive Shifts	>1 min/GB	<10 sec/GB	6-12x
Online Archives	<10K files	>50K files	5-10x
Backup Speed	>5 min/GB	<1 min/GB	5-10x
Offline Reprocessing	>15 min/GB	30 sec/GB	30x

PI AFSDK 2012 Rich Data Access

- Native Support for .Net applications
- No COM layer
- Free threading
- Our Investment is going into AFSDK





What does this imply about the PISDK?

The **OSIsoft.AF.PI** namespace provides a set of classes that can be used to manage connections and access information about a PI Data Archive.

Classes

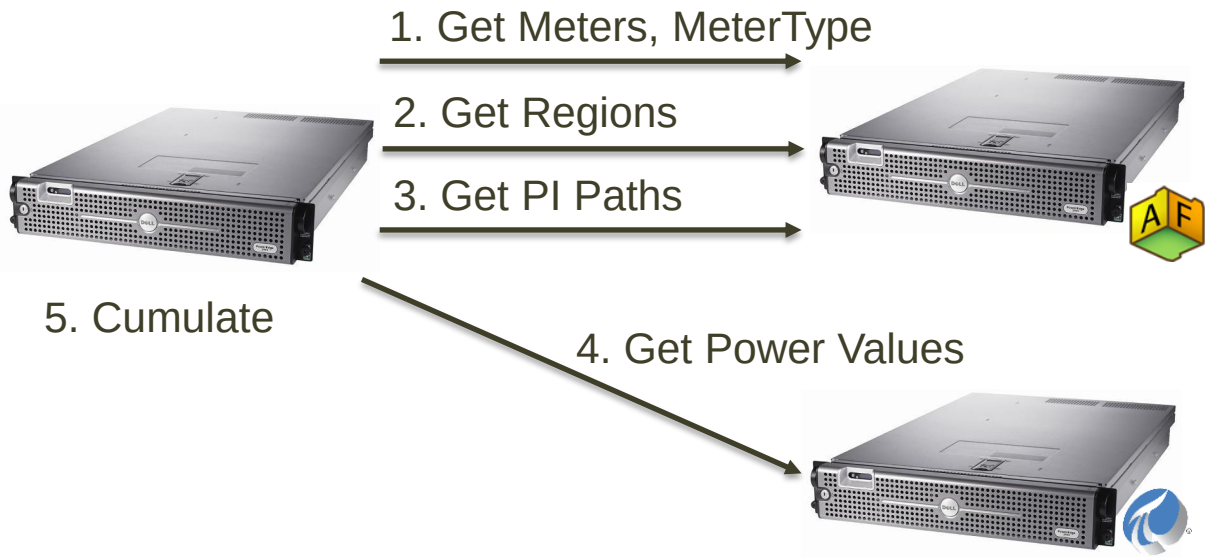
	Class	Description
	PICollective	The PICollective object is used to provide the information and status about the PIServer collective.
	PICollectiveMember	The PICollectiveMember object is used to provide the information about the member server defined within the PICollective.
	PICollectiveMembers	A collection of PICollectiveMember objects maintained by the PICollective.
	PICommonPointAttributes	This class contains information about common PIPoint attributes.
	PIConfigurationException	Represents a configuration error reading the list of configured PIServers.
	PIConnectFailedEventArgs	Privides data for the ConnectFailed event.
	PIConnectionException	Represents a connection error attempting to communicate with the specified PIServer.
	PIConnectionInfo	The PIConnectionInfo object is used to provide the connection configuration information and status to the PIServer.
	PIDataPipe	The **PIDataPipe** is used to subscribe for events on a list of PIPoint instances. The PIPoint instances do not have to be associated with the same PIServer. **PIDataPipe** instances can sign up for archive or snapshot events. This is specified when the object is constructed.
	PIException	Represents an error returned from the PIServer.
	PIPoint	The PIPoint object represents a point on a PIServer.
	PIPointClass	The PIPointClass object represents a point class on a PIServer.
	PIPointClasses	The PIPointClasses collection represents the available PIPointClass objects on a particular PIServer.
	PIPointInvalidException	The PIPoint is no longer valid on the PIServer. Most likely, the point is deleted on the server.
	PIPointList	The PIPointList object represents a list of PIPoint objects. The PIPoints can be on different PIServer. PIPointList supports duplicate PIPoint in the list.
	PISerializationException	Represents a serialization error in data returned from the PIServer.
	PIServer	The PIServer object represents a single PI Data Archive.
	PIServers	The global collection of PIServer objects maintained by the PI Server directory service represents the known PI Servers available from a workstation for client applications.
	PITimeoutException	Represents a timeout error returned from the PIServer.

PI AFSDK 2012 Rich Data Access

Capability	PIPoint	PIPointList	AFAttribute / AFData	AFAttributeList / AFListData
Snapshot	Yes	Yes	Yes (AFAttribute.GetValue)	Yes (AFAttributeList)
Recorded Value	Yes	Yes	Yes	Yes
Interpolated Value	Yes	Yes	Yes	Yes
Summary	Yes	No	Yes	No
Recorded Values	Yes	No	Yes	No
Interpolated Values	Yes	No	Yes	No
Plot Values	Yes	No	Yes	No
Summaries	Yes	No	Yes	No
Filtered Summaries	Yes	No	Yes	No
Annotations	Yes	No	Yes (PIPoint DR only)	No
Update Value	Yes	n/a	Yes	n/a
Update Values	Yes	Yes (PIServer.UpdateValues)	Yes	Yes
Calculated Values	Yes (AFCalculation)	No	Yes (AFCalculation)	No
Data Pipe	Yes (PIDataPipe)	Yes (PIDataPipe)	No	No

Example 1: A difficult query

Aggregate the power usage by region of residential meters over the last month



Name	MeterType	Region	Power
test Elements 1 2 3 4 5 6 7 8 9 10 11 ... Next			
Meter_00000001	Industrial	Mid Atlantic	87.35986 kW
Meter_00000002	Industrial	Mountain	34.00842 kW
Meter_00000003	Residential	Northwest	39.43348 kW
Meter_00000004	Industrial	Gulf Coast	20.59793 kW
Meter_00000005	Residential	Gulf Coast	1.986161 kW
Meter_00000006	Residential	Northwest	82.64814 kW
Meter_00000007	Residential	Northwest	99.97608 kW
Meter_00000008	Industrial	Mid Atlantic	12.64049 kW
Meter_00000009	Industrial	Mountain	56.1795 kW
Meter_00000010	Residential	Mid Atlantic	71.04798 kW
Meter_00000011	Industrial	Mid Atlantic	13.85982 kW
Meter_00000012	Residential	Northeast	6.343781 kW
Meter_00000013	Industrial	Great Plains	24.16104 kW
Meter_00000014	Residential	Midwest	39.29282 kW
Meter_00000015	Residential	Mid Atlantic	38.8597 kW
Meter_00000016	Industrial	Midwest	1.372978 kW
Meter_00000017	Residential	Northeast	31.87811 kW

5000 Residential, 5000 Industrial

Most Straightforward Approach

```
Dictionary<string, double> aggregates = new Dictionary<string, double>();

foreach (AFElement element in testDB.Elements)
{
    if ((element.Template != null) && (element.Template.Name == "Meter"))
    {
        string meterType = (string)element.Attributes["MeterType"].GetValue().Value;
        if (meterType == "Residential")
        {
            string region = (string)element.Attributes["Region"].GetValue().Value;
            AFValues vals = element.Attributes["Power"].Data.RecordedValues(dataRange, AFBoundaryType.Inside, null, null, false);
            double sum = AddUpValues(vals);
            AddToAggregate(aggregates, region, sum);
        }
    }
}

return aggregates;
```

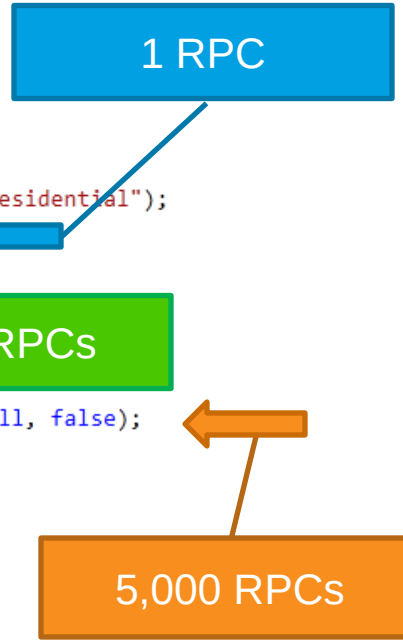
10,000 RPCs

5,000 RPCs

Time = 60 s

Search on the Server, Not the Client

```
Dictionary<string, double> aggregates = new Dictionary<string, double>();

AFElementTemplate meterTemplate = testDB.ElementTemplates["Meter"];
AFAttributeValueQuery[] query = new AFAttributeValueQuery[1];
query[0] = new AFAttributeValueQuery(meterTemplate.AttributeTemplates["MeterType"], AFSearchOperator.Equal, "Residential");
AFNamedCollectionList<AFElement> elements = AFElement.FindElementsByAttribute(null, null, query, true,
    AFSortField.Name, AFSortOrder.Ascending, maxSearch);

foreach (AFElement element in elements)
{
    string region = element.Attributes["Region"].GetValue().Value.ToString();
    AFValues vals = element.Attributes["Power"].Data.RecordedValues(dataRange, AFBoundaryType.Inside, null, null, false);
    double sum = AddUpValues(vals);
    AddToAggregate(aggregates, region, sum);
}

return aggregates;
```

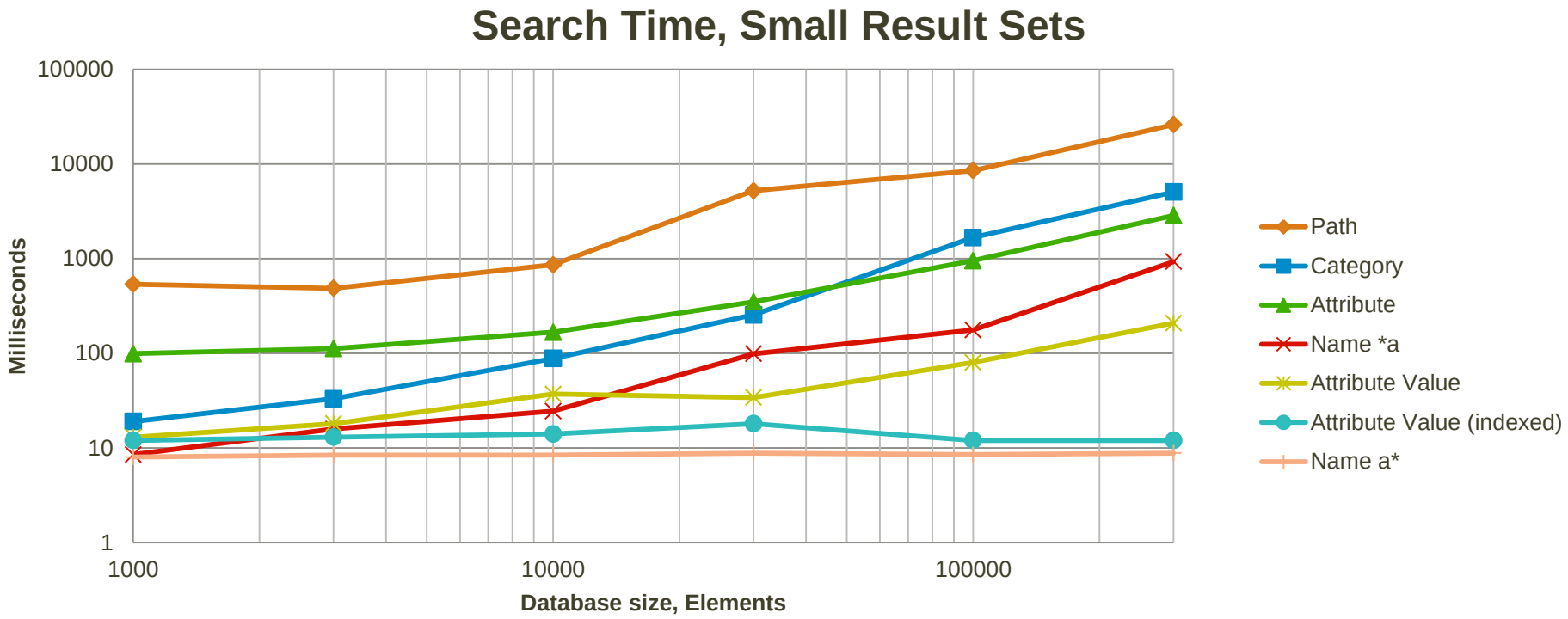


Time = 45 s

Searching in the AFSDK

Method	Returns	Name	Description	Search Root	Template	Category	Reference Type	Element Type	Paths	Attribute Value	Relative Object	Max Bandwidth	Caveats
FindElementsByPath	Element								X		X	1000 /sec	Portions execute on the client
FindElements	Element	X		X	X	X		X				20k /sec	
FindElements	Element	X	X	X	X	X						20k /sec	
FindElementsByAttribute	Element	X		X						X		20k /sec	Index attributes for best performance
FindElementsByTemplate	Element			X	X							20k /sec	
FindElementsByCategory	Element			X		X						20k /sec	
FindElemenstByReferenceType	Element			X			X					20k /sec	
FindElementAttributes	Attribute	X		X	X	X		X				20k /sec	
FindAttributesByPath	Attribute								X		X	1000 /sec	Portions execute on the client

Searching Metrics

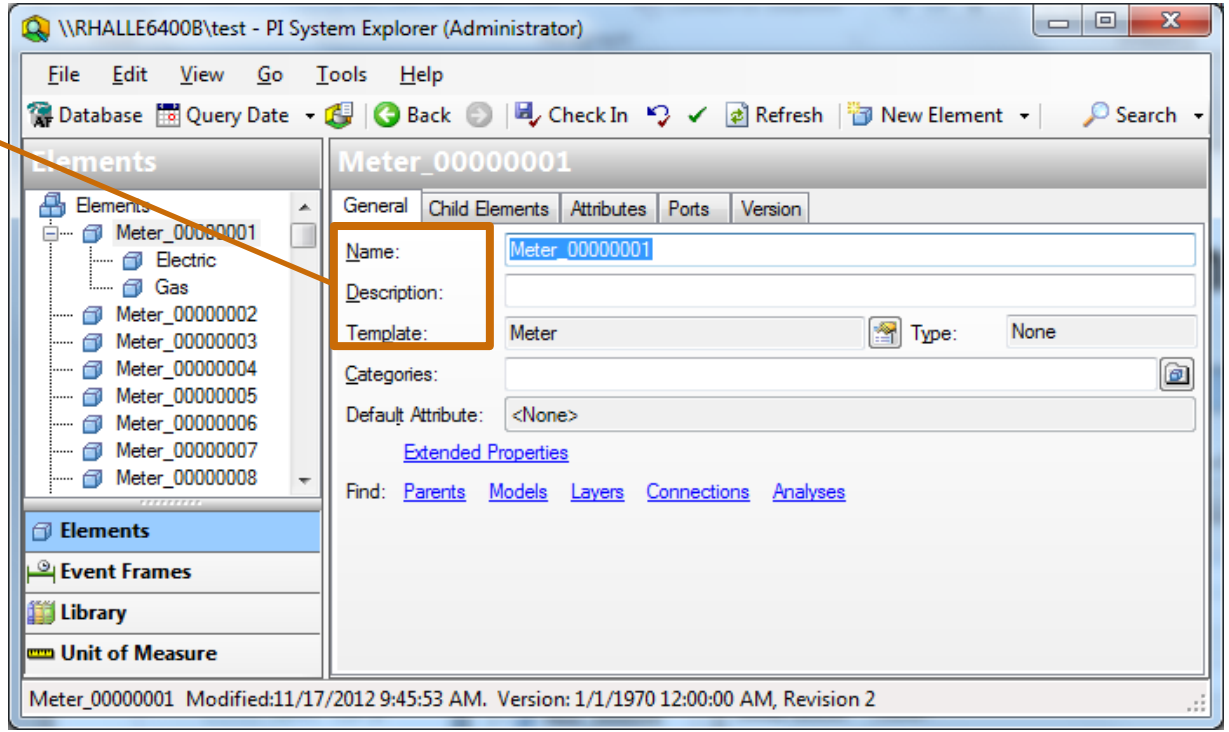


Searching Guidelines

- **Best:**
  - **Indexed Attribute Values**
    - **Even if it means changing the AF Model**
  - **Inherently Indexed Properties**
    - **Name, UniqueID**
- **Good:**
  - **Other Properties**
    - **Description, Category, Template**
  - **Attribute Values (not indexed)**
- **Caution:**
  - **Attribute Searches**
  - **Path Searches**
  - **Searches with large results sets (Limited by Bandwidth)**
  - **Complex Criteria (more joins = slower performance)**

AFElement State: Header Only

Header



AFElement State: Fully Loaded

Header

+

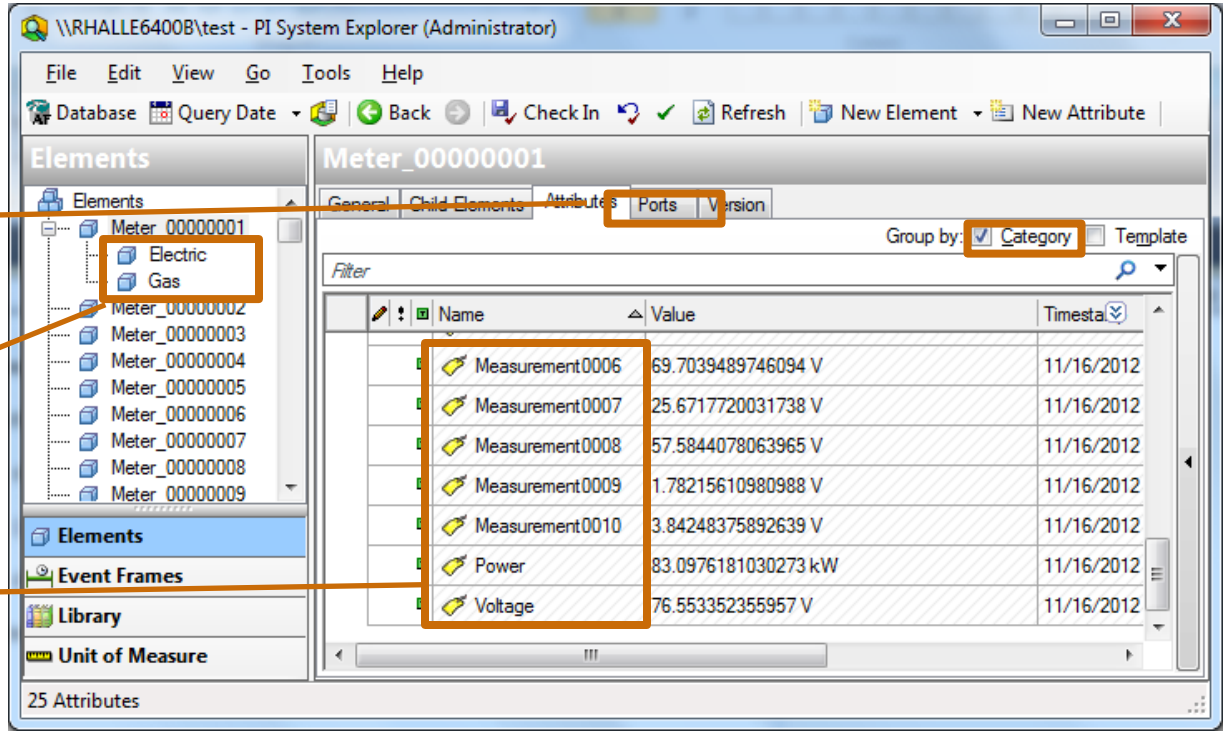
Ports, Categories, Files, Extended Properties

+

Child References

+

All Attributes



AFElement State: Partially Loaded

Header

+

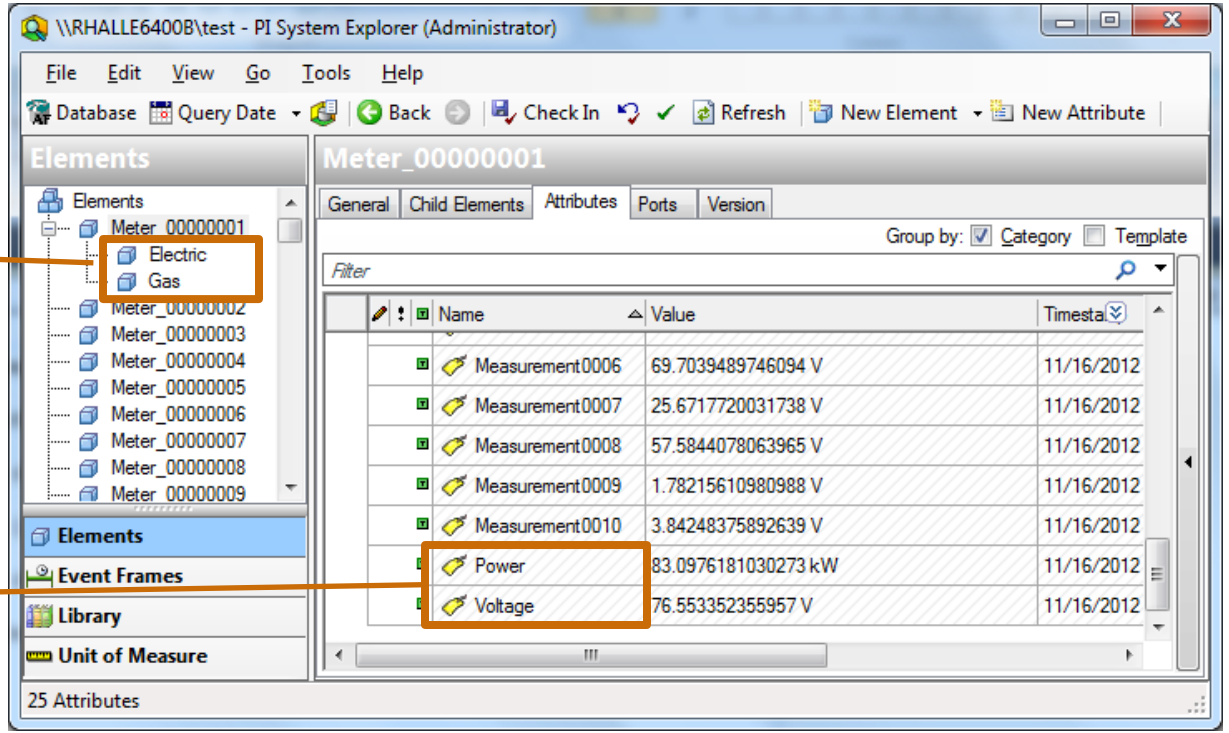
Child References

```
AFElement.LoadElementReferences(list);
```

And / Or

Selected Attributes

```
AFElement.LoadAttributes(list, attrTemplates);
```



How Lazy Loading works

- Objects are loaded on access of any portion that is not already loaded
- Collection of Headers are loaded on access

```
private static void LazyLoadExample(AFDatabase testDB)
{
    foreach (AFElement e in testDB.Elements)
    {
        if (e.Template.Name == strMeterTemplate)
        {
            string meterType = (string)e.Attributes[strMeterType].GetValue().Value;

            foreach (AFElement child in e.Elements)
            {
                string childMeterType = (string)child.Attributes[strMeterType].GetValue().Value;
            }
        }
    }

}
```

?

Collection of Headers

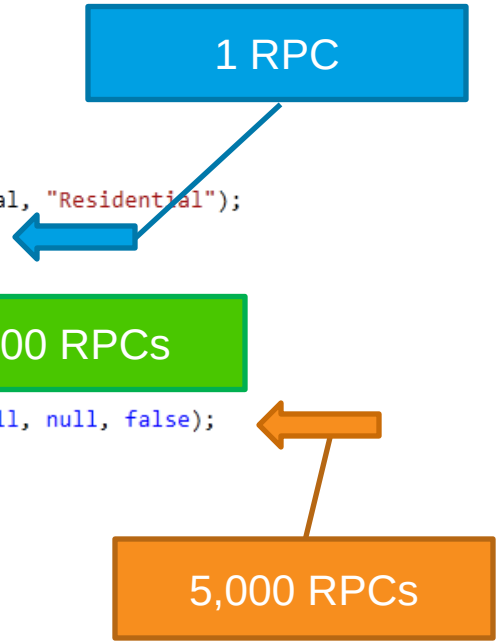
Full Element

Collection of Headers

Full Element

Complex Query Example…

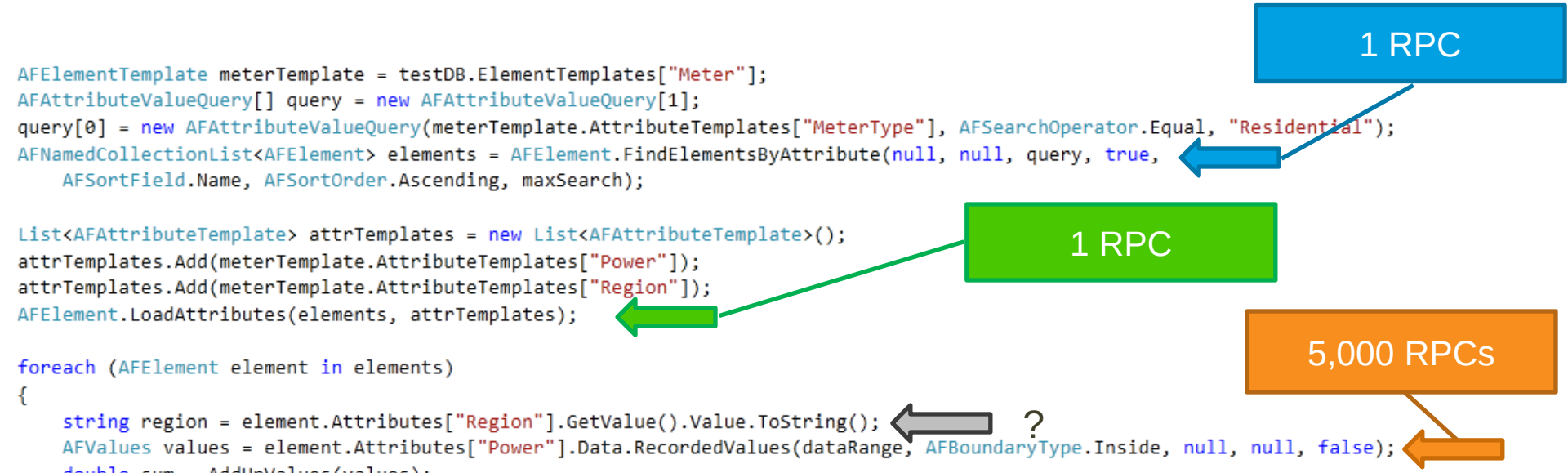
```
Dictionary<string, double> aggregates = new Dictionary<string, double>();

AFElementTemplate meterTemplate = testDB.ElementTemplates["Meter"];
AFAttributeValueQuery[] query = new AFAttributeValueQuery[1];
query[0] = new AFAttributeValueQuery(meterTemplate.AttributeTemplates["MeterType"], AFSearchOperator.Equal, "Residential");
AFNamedCollectionList<AFElement> elements = AFElement.FindElementsByAttribute(null, null, query, true,
    AFSortField.Name, AFSortOrder.Ascending, maxSearch);

foreach (AFElement element in elements)
{
    string region = element.Attributes["Region"].GetValue().Value.ToString();
    AFValues vals = element.Attributes["Power"].Data.RecordedValues(dataRange, AFBoundaryType.Inside, null, null, false);
    double sum = AddUpValues(vals);
    AddToAggregate(aggregates, region, sum);
}

return aggregates;
```



Time = 45 s

Load only the metadata you need

```
AFElementTemplate meterTemplate = testDB.ElementTemplates["Meter"];
AFAttributeValueQuery[] query = new AFAttributeValueQuery[1];
query[0] = new AFAttributeValueQuery(meterTemplate.AttributeTemplates["MeterType"], AFSearchOperator.Equal, "Residential");
AFNamedCollectionList<AFElement> elements = AFElement.FindElementsByAttribute(null, null, query, true,
    AFSortField.Name, AFSortOrder.Ascending, maxSearch);

List<AFAttributeTemplate> attrTemplates = new List<AFAttributeTemplate>();
attrTemplates.Add(meterTemplate.AttributeTemplates["Power"]);
attrTemplates.Add(meterTemplate.AttributeTemplates["Region"]);
AFElement.LoadAttributes(elements, attrTemplates);

foreach (AFElement element in elements)
{
    string region = element.Attributes["Region"].GetValue().Value.ToString();
    AFValues values = element.Attributes["Power"].Data.RecordedValues(dataRange, AFBoundaryType.Inside, null, null, false);
    double sum = AddUpValues(values);
    AddToAggregate(aggregates, region, sum);
}
```

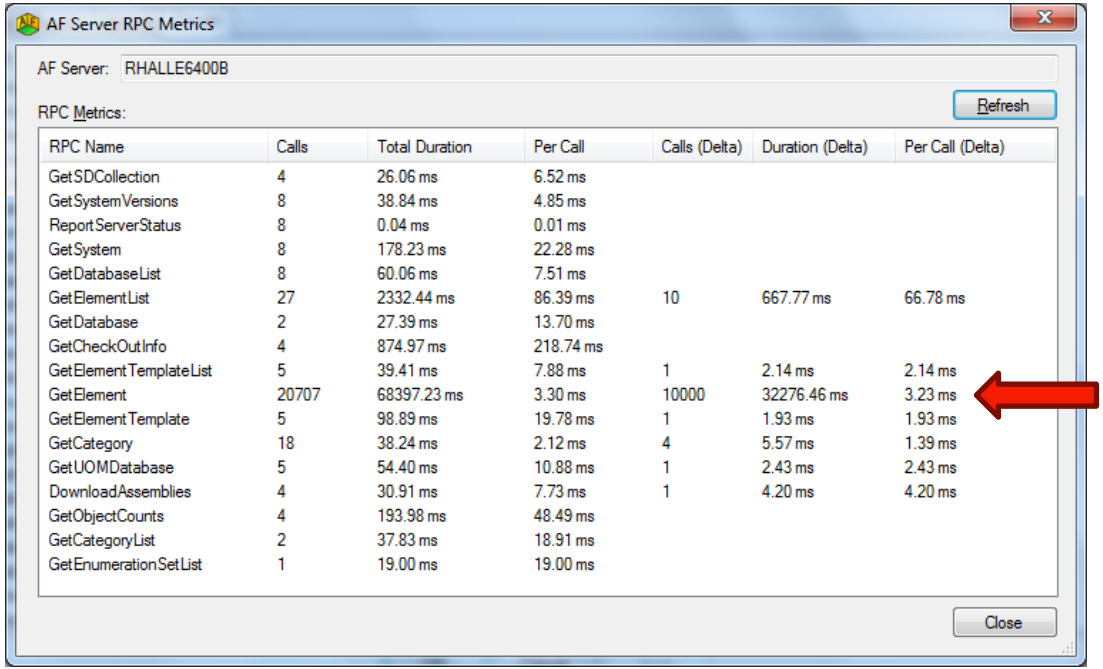


Time = 28 s

Getting Attributes – How to Diagnose

```
C:\program files\pipc\af\afservice.exe.config
<!-- Indicates that all RPC entries into the AF Application Server should be logged. (for Troubleshooting) -->
<add key="logRPCDurations" value="True" />
```

In PI System Explorer:
File
AF Server Properties
Counts
Right Click
RPC Metrics:



RPC Name	Calls	Total Duration	Per Call	Calls (Delta)	Duration (Delta)	Per Call (Delta)
GetSDCollection	4	26.06 ms	6.52 ms			
GetSystemVersions	8	38.84 ms	4.85 ms			
ReportServerStatus	8	0.04 ms	0.01 ms			
GetSystem	8	178.23 ms	22.28 ms			
GetDatabaseList	8	60.06 ms	7.51 ms			
GetElementList	27	2332.44 ms	86.39 ms	10	667.77 ms	66.78 ms
GetDatabase	2	27.39 ms	13.70 ms			
GetCheckOutInfo	4	874.97 ms	218.74 ms			
GetElementTemplateList	5	39.41 ms	7.88 ms	1	2.14 ms	2.14 ms
GetElement	20707	68397.23 ms	3.30 ms	10000	32276.46 ms	3.23 ms
GetElementTemplate	5	98.89 ms	19.78 ms	1	1.93 ms	1.93 ms
GetCategory	18	38.24 ms	2.12 ms	4	5.57 ms	1.39 ms
GetUOMDatabase	5	54.40 ms	10.88 ms	1	2.43 ms	2.43 ms
DownloadAssemblies	4	30.91 ms	7.73 ms	1	4.20 ms	4.20 ms
GetObjectCounts	4	193.98 ms	48.49 ms			
GetCategoryList	2	37.83 ms	18.91 ms			
GetEnumerationSetList	1	19.00 ms	19.00 ms			

```
private static void ParallelLoadAttributes(IList<AFElement> elements, IList<AFAttributeTemplate> attributeTemplates, int nThreads)
{
    Thread[] th = new Thread[nThreads];
    List<AFElement>[] lists = new List<AFElement>[nThreads];
    LoadAttributesParms[] o = new LoadAttributesParms[nThreads];

    for (int j = 0; j < nThreads; j++)
    {
        lists[j] = new List<AFElement>();
        o[j] = new LoadAttributesParms();
    }

    for (int j = 0; j < elements.Count; j++)
        lists[j % nThreads].Add(elements[j]);

    for (int j = 0; j < nThreads; j++)
    {
        o[j].attributeTemplates = attributeTemplates;
        o[j].elements = lists[j];
        th[j] = new Thread(LoadAttributesProc);
        th[j].Start((object)o[j]);
    }

    for (int j = 0; j < nThreads; j++)
        th[j].Join();
}

static private void LoadAttributesProc(object o)
{
    AFElement.LoadAttributes(((LoadAttributesParms)o).elements, ((LoadAttributesParms)o).attributeTemplates);
}
```

Various Approaches for Loading Metadata

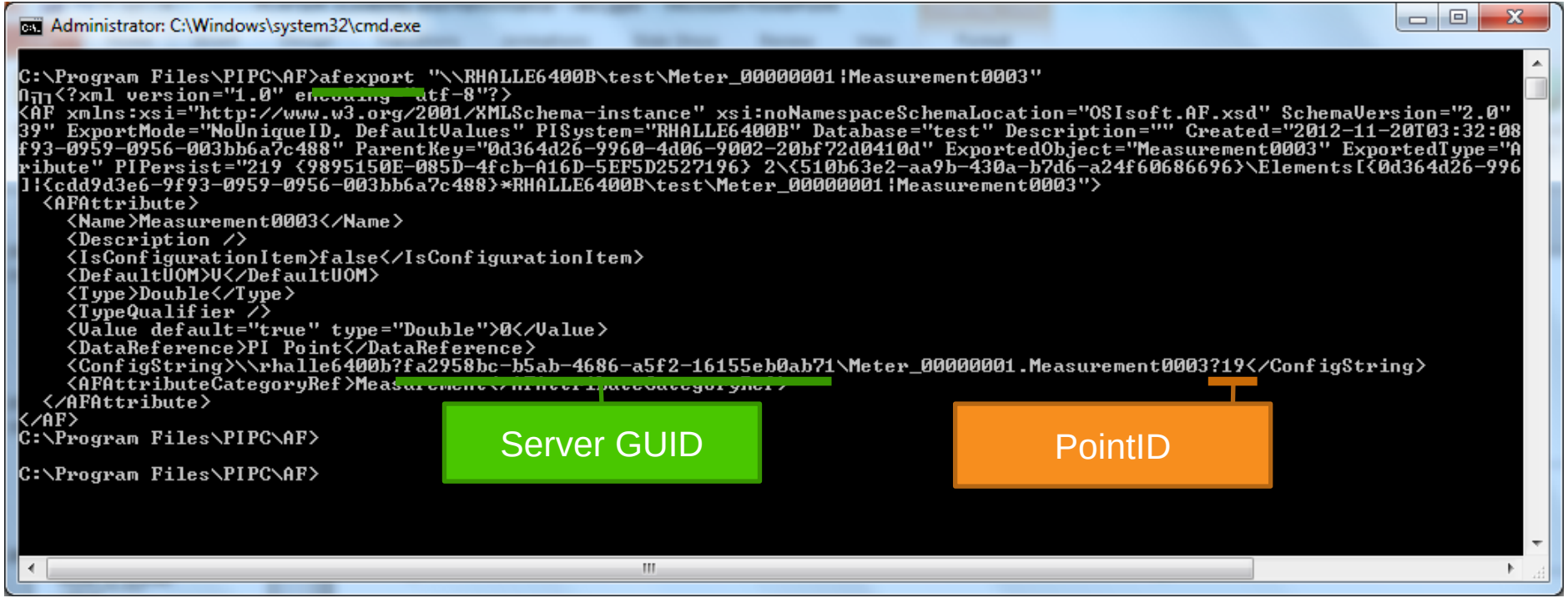
Find Method	Load Method	Time (s)
Search on Client	Lazy Load	31.4
Attribute Search	Lazy Load	16.1
Attribute Search	Load Elements	6.4
Attribute Search	Load Attributes	1.0
Attribute Search	Parallel Load Attributes	0.7

Time = 27.7 s

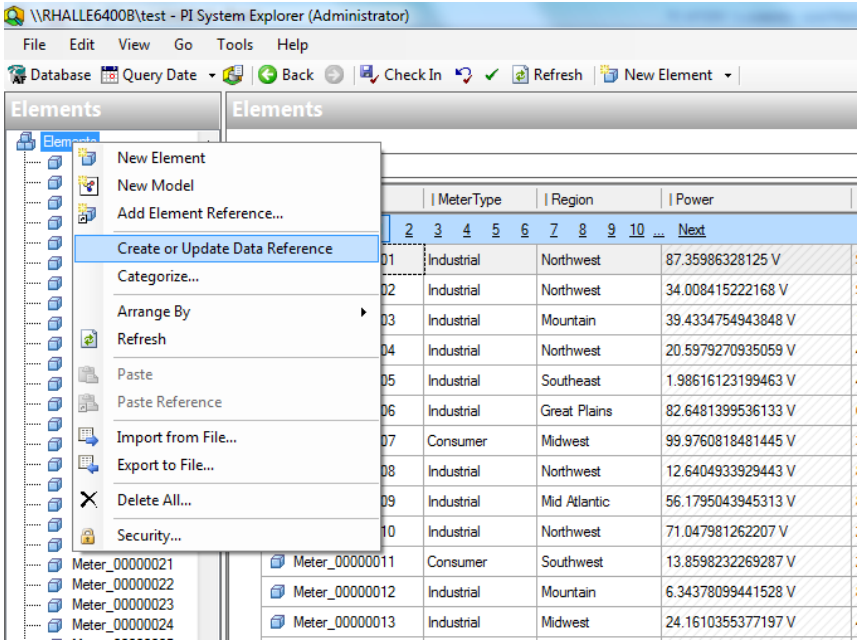
Point Path Resolution

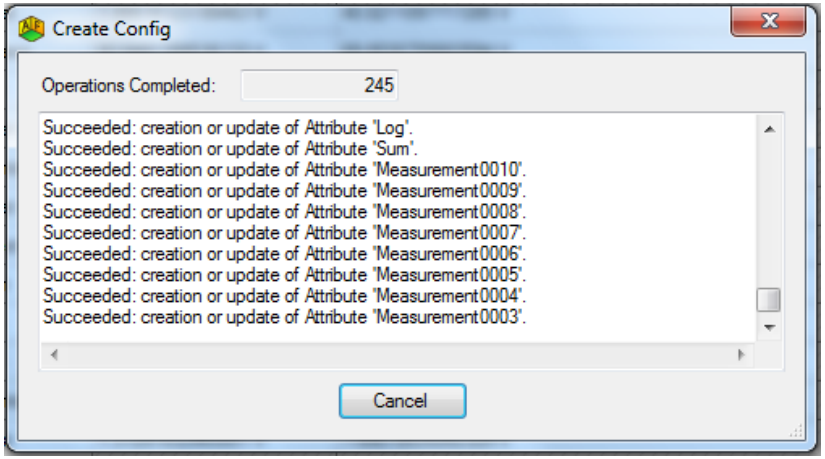
- Calculated from the template:
  - \\%Server%\%Element%.%Attribute%
- Unresolved:
  - \\rhalle6400b\Meter_00000001.Measurement0001
- Resolved:
  - \\rhalle6400b?fa2958bc-b5ab-4686-a5f2-16155eb0ab71\Meter_00000001.Measurement0001?17

Point Path Resolution- How do you Know?



Point Path Resolution





Time = 26.6 s

Getting Data – Bulk if Possible

Capability	PIPoint	PIPointList	AFAttribute / AFData	AFAttributeList / AFListData
Snapshot	Yes	Yes	Yes (AFAttribute.GetValue)	Yes
Recorded Value	Yes	Yes	Yes	Yes
Interpolated Value	Yes	Yes	Yes	Yes
Summary	Yes	No	Yes	No
Recorded Values	Yes	No	Yes	No
Interpolated Values	Yes	No	Yes	No
Plot Values	Yes	No	Yes	No
Summaries	Yes	No	Yes	No
Filtered Summaries	Yes	No	Yes	No
Annotations	Yes	No	Yes (PIPoint DR only)	No
Update Value	Yes	n/a	Yes	n/a
Update Values	Yes	Yes (PIServer.UpdateValues)	Yes	Yes
Calculated Values	Yes (AFCalculation)	No	Yes (AFCalculation)	No
Data Pipe	Yes (PIDataPipe)	Yes (PIDataPipe)	No	No

Getting Data – Parallel

```
private static Dictionary<string, double> ParallelAttributeValues(List<AFElement> elementList, AFTimeRange tr, int nThreads)
{
    Thread[] th = new Thread[nThreads];
    List<AFElement>[] lists = new List<AFElement>[nThreads];
    ParallelAttributeParms[] o = new ParallelAttributeParms[nThreads];
    Dictionary<string, double> results = new Dictionary<string, double>();

    for (int j = 0; j < nThreads; j++)
    {
        lists[j] = new List<AFElement>();
        o[j] = new ParallelAttributeParms();
        o[j].tr = tr;
        o[j].results = new Dictionary<string, double>();
    }

    for (int j = 0; j < elementList.Count; j++)
    {
        int k = j % nThreads;
        lists[k].Add(elementList[j]);
    }

    for (int j = 0; j < nThreads; j++)
    {
        o[j].elements = lists[j];
        th[j] = new Thread(ParallelAttributeValuesProc);
        th[j].Start((object)o[j]);
    }

    for (int j = 0; j < nThreads; j++)
    {
        th[j].Join();
        foreach (string key in o[j].results.Keys)
            AddToAggregate(results, key, o[j].results[key]);
    }
    return results;
}
```

Getting Data – Parallel

```
private static Dictionary<string, double> ComplexQuery5(AFDatabase testDB, AFTimeRange dataRange, int nThreads)
{
    Dictionary<string, double> aggregates;

    AFElementTemplate meterTemplate = testDB.ElementTemplates["Meter"];
    AFAttributeValueQuery[] query = new AFAttributeValueQuery[1];
    query[0] = new AFAttributeValueQuery(meterTemplate.AttributeTemplates["MeterType"], AFSearchOperator.Equal, "Residential");
    AFNamedCollectionList<AFElement> elements = AFElement.FindElementsByAttribute(null, null, query, true, AFSortField.Name,
        AFSortOrder.Ascending, maxSearch);

    IList<AFAttributeTemplate> attributeTemplates = new List<AFAttributeTemplate>();
    attributeTemplates.Add(meterTemplate.AttributeTemplates["Power"]);
    attributeTemplates.Add(meterTemplate.AttributeTemplates["Region"]);
    ParallelLoadAttributes(elements, attributeTemplates, optimalSqlThreads);

    aggregates = ParallelAttributeValues(elements.ToList<AFElement>(), dataRange, nThreads);

    return aggregates;
}

static private void ParallelAttributeValuesProc(object o)
{
    foreach (AFElement e in ((ParallelAttributeParms)o).elements)
    {
        string region = e.Attributes["Region"].GetValue().Value.ToString();
        AFValues vals = e.Attributes["Power"].Data.RecordedValues(((ParallelAttributeParms)o).tr, AFBoundaryType.Inside, null, null, false);
        double sum = AddUpValues(vals);
        AddToAggregate(((ParallelAttributeParms)o).results, region, sum);
    }
}
```

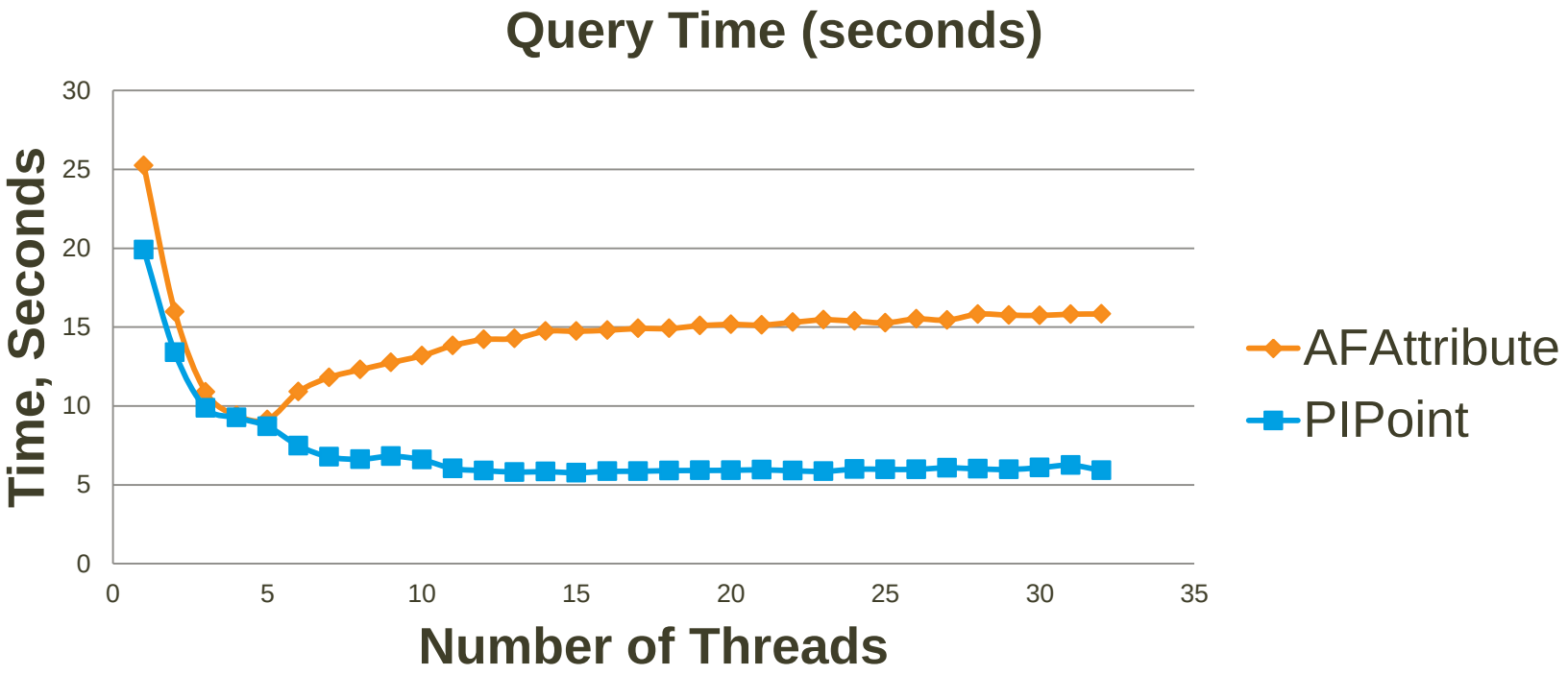
1 RPC

1 RPC

5,000 RPCs
6 Threads

Time = 9.2 s

Why Does Parallel Work?



Getting Data – PIPoint vs. AFAttribute

```
static private void ParallelAttributeValuesProc(object o)
{
    foreach (AFElement e in ((ParallelAttributeParms)o).elements)
    {
        string region = e.Attributes["Region"].GetValue().Value.ToString();
        AFValues vals = e.Attributes["Power"].Data.RecordedValues(((ParallelAttributeParms)o).tr, AF                ype.Inside, null, null, false);
        double sum = AddUpValues(vals);
        AddToAggregate(((ParallelAttributeParms)o).results, region, sum);
    }
}
```

5,000 RPCs
6 Threads

```
static private void ParallelPointValuesProc(object o)
{
    foreach (AFElement e in ((ParallelAttributeParms)o).elements)
    {
        string region = e.Attributes["Region"].GetValue().Value.ToString();
        AFValues vals = e.Attributes["Power"].PIPoint.RecordedValues(((ParallelAttributeParms)o).tr, A                ype.Inside, null, false);
        double sum = AddUpValues(vals);
        AddToAggregate(((ParallelAttributeParms)o).results, region, sum);
    }
}
```

5000 RPCs
24 Threads

Time = 6.0 s

Getting Data – Summarize on the Server

```
    List<AFAttribute> attrList = new List<AFAttribute>();
    foreach (AFElement element in elements)
        attrList.Add(element.Attributes["Power"]);

    List<AFValue> vals = ParallelSummary(attrList, dataRange, AFSummaryTypes.Total,
        AFCalculationBasis.EventWeighted, AFTimestampCalculation.Auto, optimalPIPointThreads);

    foreach (AFValue val in vals)
    {
        string region = (string)val.Attribute.Element.Attributes["Region"].GetValue().Value;
        AddAFValueToAggregate(aggregates, val, region);
    }

    return aggregates;
}

static private void SummaryProc(object o)
{
    foreach (AFAttribute at in ((SummaryParms)o).attributes)
    {
        IDictionary<AFSummaryTypes, AFValue> vals = at.PIPoint.Summary(((SummaryParms)o).tr, ((SummaryParms)o).summaryType,
            ((SummaryParms)o).calculationBasis, ((SummaryParms)o).timestampCalculation);
        AFValue val = vals[((SummaryParms)o).summaryType];
        val.Attribute = at;
        ((SummaryParms)o).results.Add(val);
    }
}
```

5,000 RPCs
24 Threads

Time = 1.8 s

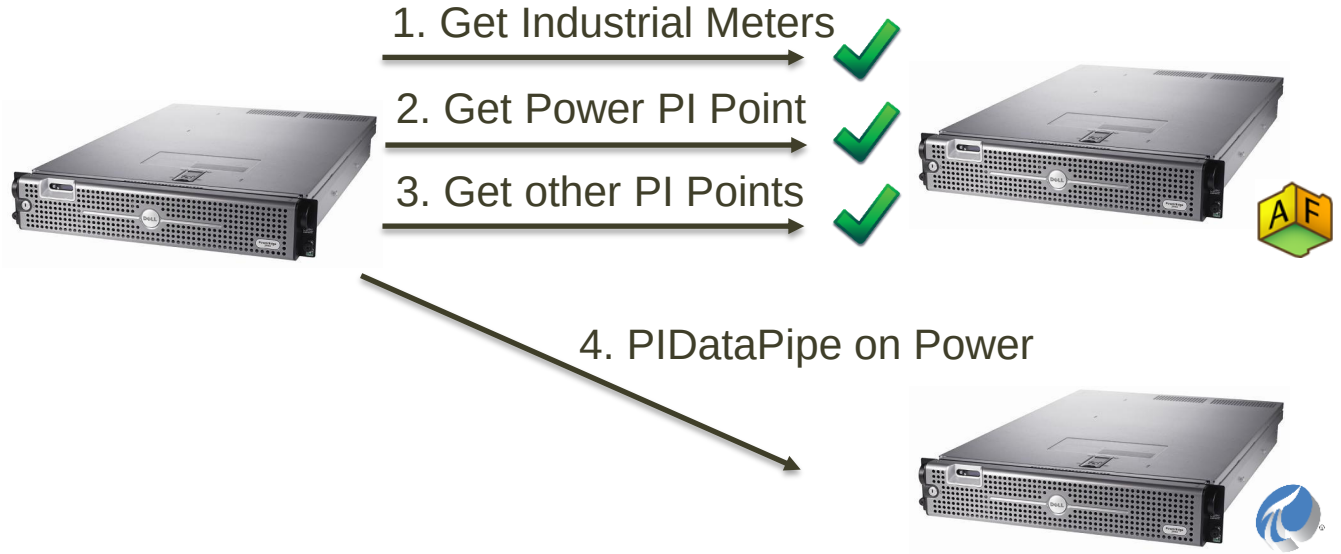
Options for Calculating on the Server

Capability	PIPoint	PIPointList	AFAttribute / AFData	AFAttributeList / AFListData
Snapshot	Yes	Yes	Yes (AFAttribute.GetValue)	Yes
Recorded Value	Yes	Yes	Yes	Yes
Interpolated Value	Yes	Yes	Yes	Yes
Summary	Yes	No	Yes	No
Recorded Values	Yes	No	Yes	No
Interpolated Values	Yes	No	Yes	No
Plot Values	Yes	No	Yes	No
Summaries	Yes	No	Yes	No
Filtered Summaries	Yes	No	Yes	No
Annotations	Yes	No	Yes (PIPoint DR only)	No
Update Value	Yes		Yes	
Update Values	Yes	Yes (PIServer.UpdateValues)	Yes	Yes
Calculated Values	Yes (AFCalculation)	No	Yes (AFCalculation)	No
Data Pipe	Yes (PIDataPipe)	Yes (PIDataPipe)	No	No

Example 2: Streaming Calculations

As data streams in, calculate Power Factor as Power (kw) /Apparent Power for Industrial Meters

Part 1: Setup



Streaming Calculations, Setup

```
private static void FindMetersAndSignUp()
{
    AFElementTemplate meterTemplate = db.ElementTemplates["Meter"];
    AFAttributeValueQuery[] query = new AFAttributeValueQuery[1];
    query[0] = new AFAttributeValueQuery(meterTemplate.AttributeTemplates["MeterType"], AFSearchOperator.Equal, "Industrial");
    AFNamedCollectionList<AFElement> elements = AFElement.FindElementsByAttribute(null, null, query, true,
        AFSortField.Name, AFSortOrder.Ascending, maxSearch);

    List<AFAttributeTemplate> attrTemplates = new List<AFAttributeTemplate>();
    attrTemplates.Add(meterTemplate.AttributeTemplates["Power"]);
    attrTemplates.Add(meterTemplate.AttributeTemplates["ApparentPower"]);
    attrTemplates.Add(meterTemplate.AttributeTemplates["PowerFactor"]);
    AFElement.LoadAttributes(elements, attrTemplates);

    pointList = new PIPointList();
    mapToAttribute = new Dictionary<PIPoint, AFAttribute>();
    foreach (AFElement element in elements)
    {
        AFAttribute powerAttribute = element.Attributes["Power"];
        PIPoint point = powerAttribute.PIPoint;
        pointList.Add(point);
        mapToAttribute.Add(point, powerAttribute);
    }

    powerEvents = new PIDataPipe(AFDataPipeType.Snapshot);
    powerEvents.AddSignups(pointList);
}
```

Search

Load Attributes

Save Element Reference

Signup in Bulk

Time = 1.4 s

Example 2: Streaming Calculations

As data streams in, calculate Power Factor as Power (kw) /Apparent Power for Industrial Meters

Part 2: Running

1. Get Power Events
2. Get Apparent Power

3. Write Power Factor

Most Straightforward Approach

```
private static int CalculatePowerFactor1()
{
    AFListResults<PIPoint, AFDataPipeEvent> events = powerEvents.GetUpdateEvents(1000);

    foreach (AFDataPipeEvent snapshotEvent in events.Results)
    {
        if (snapshotEvent.Action == AFDataPipeAction.Add || snapshotEvent.Action == AFDataPipeAction.Update)
        {
            PIPoint point = snapshotEvent.Value.PIPoint;
            AFAttribute powerAttribute = mapToAttribute[point];

            AFAttribute apparentPowerAttribute = powerAttribute.Element.Attributes["ApparentPower"];
            AFAttribute powerFactorAttribute = powerAttribute.Element.Attributes["PowerFactor"];
            float apparentPower = (float)apparentPowerAttribute.GetValue().Value;
            float power = (float)snapshotEvent.Value.Value;
            float powerFactorValue = power / apparentPower;
            AFValue powerFactor = new AFValue(powerFactorValue, snapshotEvent.Value.Timestamp);
            powerFactorAttribute.Data.UpdateValue(new AFValue(powerFactor), AFUpdateOption.Insert);
        }
    }
    return events.Count;
}
```

5 RPCs

5000 RPCs

5000 RPCs

2800 Calculations / s

Getting Data – Bulk if Possible

Capability	PIPoint	PIPointList	AFAttribute / AFData	AFAttributeList / AFListData
Snapshot	Yes	Yes	Yes (AFAttribute.GetValue)	Yes
Recorded Value	Yes	Yes	Yes	Yes
Interpolated Value	Yes	Yes	Yes	Yes
Summary	Yes	No	Yes	No
Recorded Values	Yes	No	Yes	No
Interpolated Values	Yes	No	Yes	No
Plot Values	Yes	No	Yes	No
Summaries	Yes	No	Yes	No
Filtered Summaries	Yes	No	Yes	No
Annotations	Yes	No	Yes (PIPoint DR only)	No
Update Value	Yes	n/a	Yes	n/a
Update Values	Yes	Yes (PIServer.UpdateValues)	Yes	Yes
Calculated Values	Yes (AFCalculation)	No	Yes (AFCalculation)	No
Data Pipe	Yes (PIDataPipe)	Yes (PIDataPipe)	No	No

Getting Data in Bulk Calls

```
private static int CalculatePowerFactor2()
{
    AFListResults<PIPoint, AFDataPipeEvent> events = powerEvents.GetUpdateEvents(1000);
    AFAttributeList apparentPowerList = new AFAttributeList();
    foreach (AFDataPipeEvent snapshotEvent in events.Results)
    {
        if (snapshotEvent.Action == AFDataPipeAction.Add || snapshotEvent.Action == AFDataPipeAction.Update)
        {
            PIPoint point = snapshotEvent.Value.PIPoint;
            AFAttribute powerAttribute = mapToAttribute[point];
            apparentPowerList.Add(powerAttribute.Element.Attributes["ApparentPower"]);
        }
    }

    AFValues apparentPowerValues = apparentPowerList.GetValue();
    Dictionary<AFAttribute, AFValue> mapToValues = new Dictionary<AFAttribute, AFValue>();
    foreach (AFValue val in apparentPowerValues)
        if (!mapToValues.ContainsKey(val.Attribute))
            mapToValues.Add(val.Attribute, val);
```

5 RPCs

5 RPCs

Getting Data in Bulk Calls (continued)

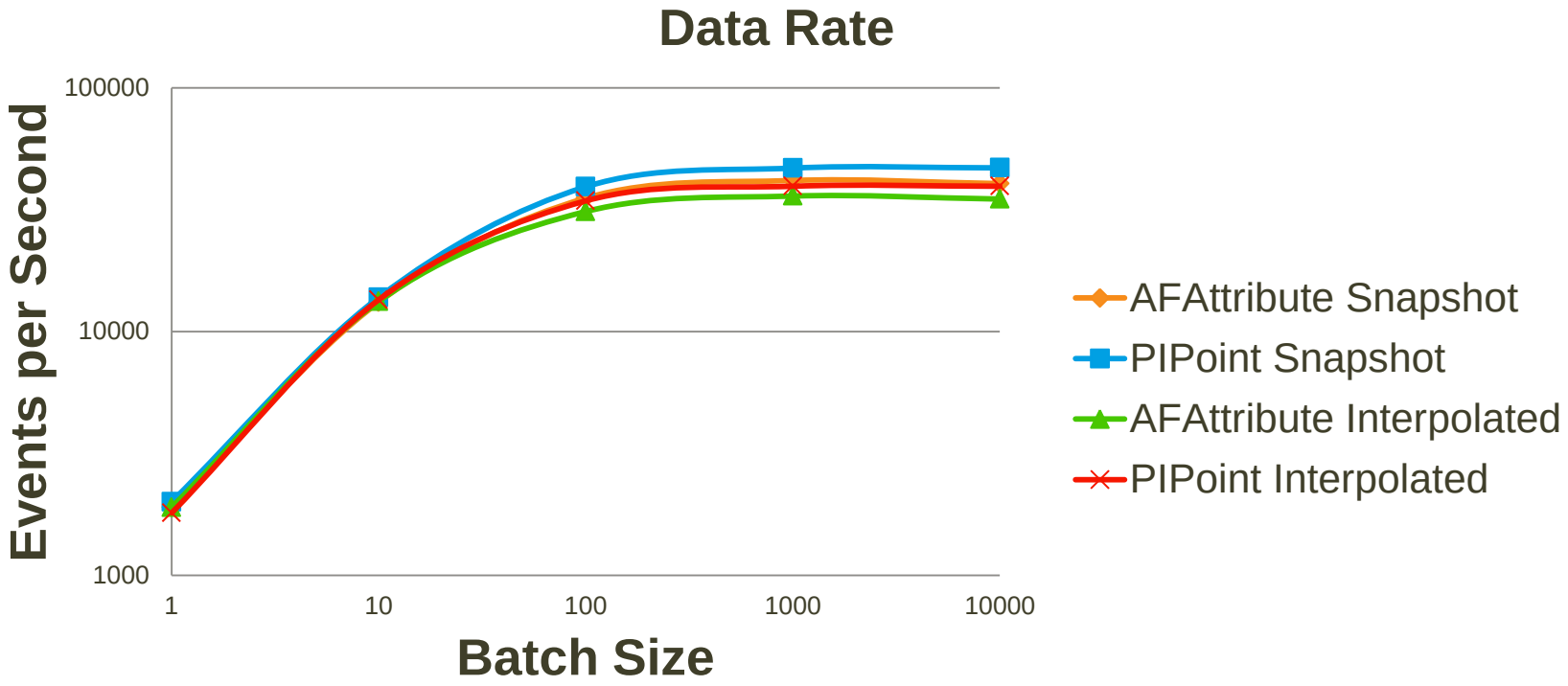
```
List<AFValue> powerFactorValues = new List<AFValue>();
foreach (AFDataPipeEvent snapshotEvent in events.Results)
{
    if (snapshotEvent.Action == AFDataPipeAction.Add || snapshotEvent.Action == AFDataPipeAction.Update)
    {
        PIPoint point = snapshotEvent.Value.PIPoint;
        AFAttribute powerAttribute = mapToAttribute[point];

        AFAttribute apparentPowerAttribute = powerAttribute.Element.Attributes["ApparentPower"];
        AFAttribute powerFactorAttribute = powerAttribute.Element.Attributes["PowerFactor"];
        float apparentPower = (float)mapToValues[apparentPowerAttribute].Value;
        float power = (float)snapshotEvent.Value.Value;
        float powerFactorValue = power / apparentPower;
        AFValue powerFactor = new AFValue(powerFactorValue, snapshotEvent.Value.Timestamp);
        powerFactorValues.Add(powerFactor);
    }
}
AFListData.UpdateValues(powerFactorValues, AFUpdateOption.NoReplace);
return events.Count;
```

Lookup the Value

5000 RPCs

4900 Calculations / s

Bulk Call Metrics

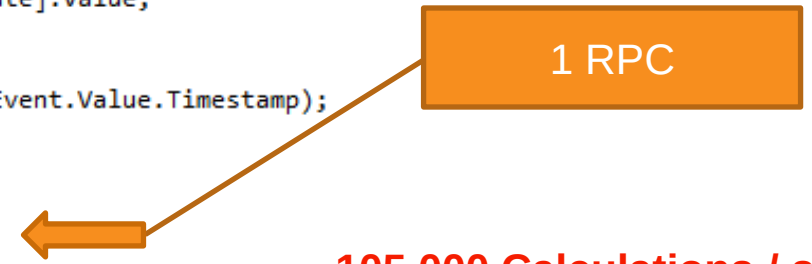


Write Data – Bulk if Possible

Capability	PIPoint	PIPointList	AFAttribute / AFData	AFAttributeList / AFListData
Snapshot	Yes	Yes	Yes (AFAttribute.GetValue)	Yes
Recorded Value	Yes	Yes	Yes	Yes
Interpolated Value	Yes	Yes	Yes	Yes
Summary	Yes	No	Yes	No
Recorded Values	Yes	No	Yes	No
Interpolated Values	Yes	No	Yes	No
Plot Values	Yes	No	Yes	No
Summaries	Yes	No	Yes	No
Filtered Summaries	Yes	No	Yes	No
Annotations	Yes	No	Yes (PIPoint DR only)	No
Update Value	Yes	n/a	Yes	n/a
Update Values	Yes	Yes (PIServer.UpdateValues)	Yes	Yes
Calculated Values	Yes (AFCalculation)	No	Yes (AFCalculation)	No
Data Pipe	Yes (PIDataPipe)	Yes (PIDataPipe)	No	No

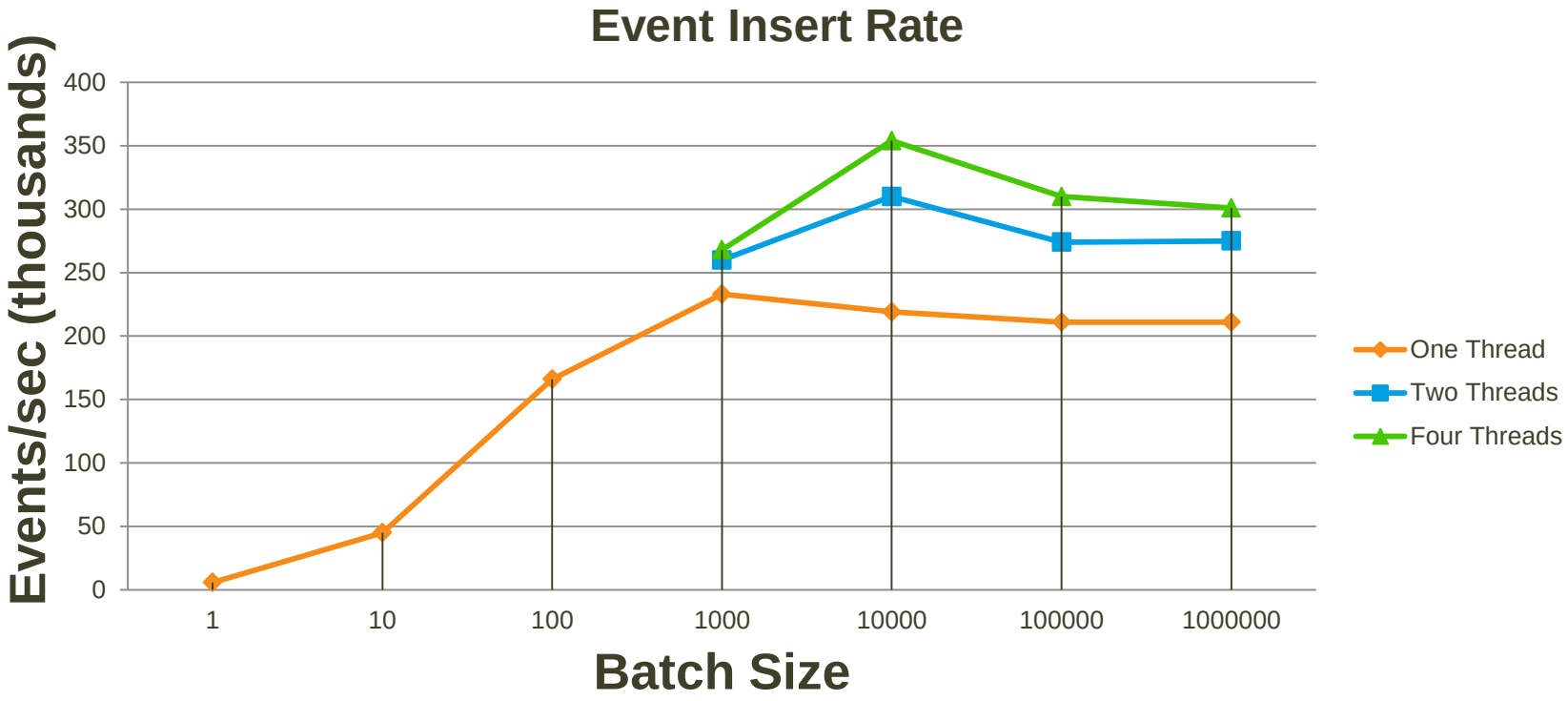
Bulk Update Events

```
List<AFValue> powerFactorValues = new List<AFValue>();
foreach (AFDataPipeEvent snapshotEvent in events.Results)
{
    if (snapshotEvent.Action == AFDataPipeAction.Add || snapshotEvent.Action == AFDataPipeAction.Update)
    {
        PIPoint point = snapshotEvent.Value.PIPoint;
        AFAttribute powerAttribute = mapToAttribute[point];

        AFAttribute apparentPowerAttribute = powerAttribute.Element.Attributes["ApparentPower"];
        AFAttribute powerFactorAttribute = powerAttribute.Element.Attributes["PowerFactor"];
        float apparentPower = (float)mapToValues[apparentPowerAttribute].Value;
        float power = (float)snapshotEvent.Value.Value;
        float powerFactorValue = power / apparentPower;
        AFValue powerFactor = new AFValue(powerFactorValue, snapshotEvent.Value.Timestamp);
        powerFactorValues.Add(powerFactor);
    }
}
AFListData.UpdateValues(powerFactorValues, AFUpdateOption.NoReplace);
return events.Count;
```



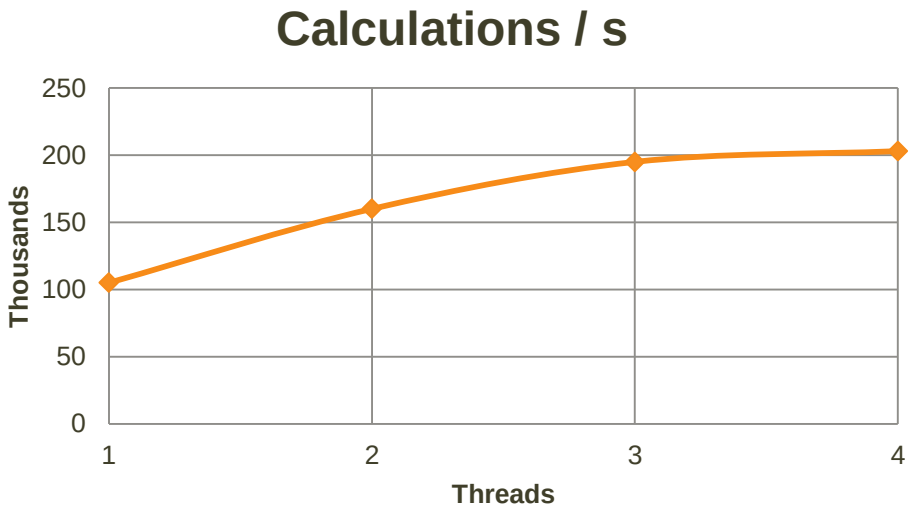
105,000 Calculations / s

Update Values Metrics



Run Calculations in Parallel

PIDataPipe supports about 100,000 events/sec *per thread*



203,000 Calculations / s

PI Data Archive Hardware Guidelines

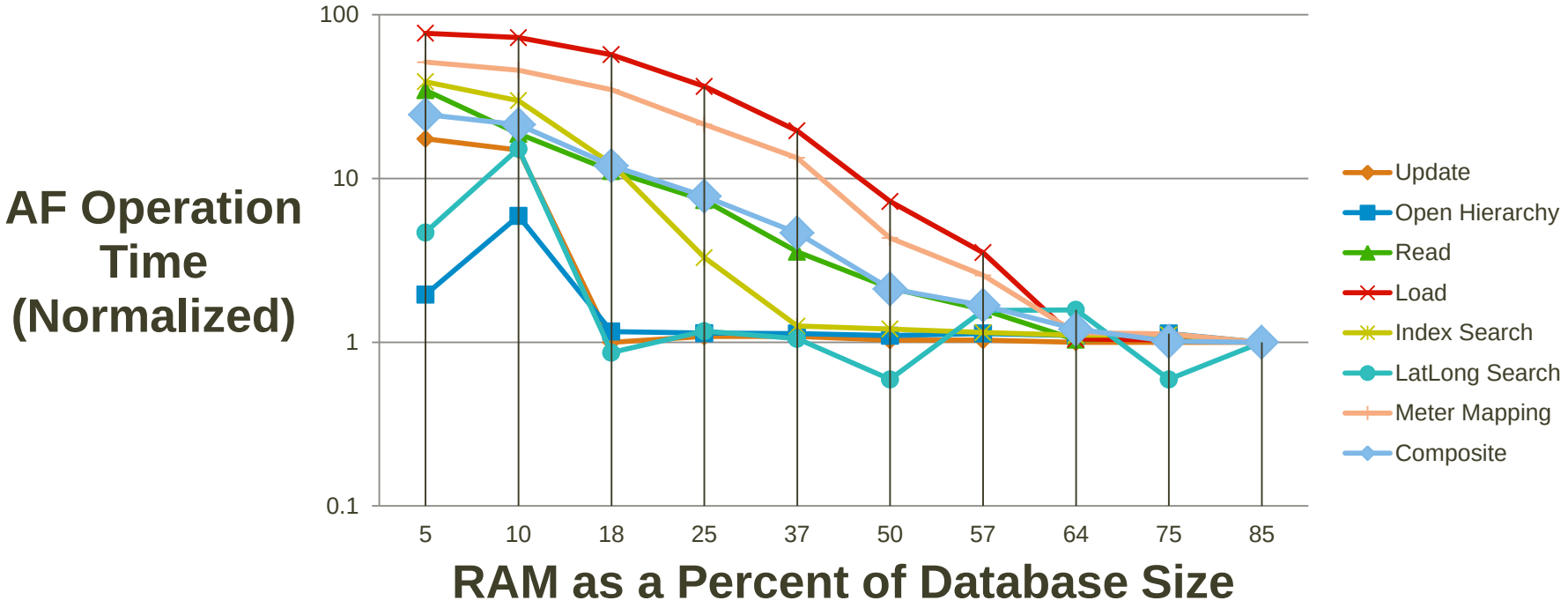
0. Windows ⇨ Latest OS (64 bits)
1. Memory ⇨ most bang/$
2. Storage ⇨ latency (IOPS)
3. Network ⇨ latency (RTT)
4. Computing ⇨ client workload

	RAM	Disk IO/s	Network	CPU Cores
Minimum	15KB per PI Point	Rate of Archived Events/50	100Mbps LAN	4 + Active Client Connections/5
Recommended	Enough to fit common/critical queries in RAM (file system cache)	Rate of Archived Events/10 + Read Workload (based on desired client response time)	1-10Gbps LAN (end-to-end latency is most critical)	4 + Active Client Connections/2 (more with multi-threaded clients)

AF Server Hardware Guidelines

1. Run the AFServer on SQL Server machine (if possible)
2. RAM for sqlserver.exe >= 60% size of PIFD
3. Drive Array Supports > 3000 random IO/s

Guidelines Matter



PI AFSDK vNext

- Buffering
- Bulk Archive Calls
- Client Cache
- AFDataPipe for AFAttribute

Summary

- Be sure your AF and PI Servers are set up correctly
- Example 1
  - Search on the server, not the client
  - Load only the metadata you need
  - Be sure that PI Point paths are resolved
  - Use PIPoint instead of AFAttribute where possible
  - Use multiple threads for PI data calls
  - Do summaries on the PI Server where possible
- Example 2
  - Read data in bulk
  - Write data in bulk
  - Use multiple threads for PIDataPipe and calculations

Ray Hall

Director, Development

OSIsoft, LLC

rhall@osisoft.com

THANK YOU