Optimizing PI AF for Performance and Scalability

Presented by Chris Manhard, AF Engineering Group Leader
Paul Combellick, AF Server Principal Engineer
Optimizing your AF SDK based Application
AF SDK Application Performance

• The fastest way to PI and AF data
• AF SDK does what you ask
• It is easy to ask inefficiently
• AF SDK does not predict your next call
• Large scale performance requires care

“With Great Power Comes Great Responsibility”
Spiderman’s Uncle Ben,
Strategies for best SDK Performance

- Bulk
- Parallel
- Process Change Streams
- Cache on the Client
- Pre-Calculate
- Diagnose Performance Bottlenecks
Make Calls in Bulk

AF Server

Pump1 Load Attributes

Pump2 Load Attributes

Pump101 Load Attributes

BA:Active Summary

Sinusoid Summary

CDT158 Summary

PI Data Archive
Bulk vs. Serial – Performance 101

Effect of Latency on Snapshot

- Server
- Latency
- Bandwidth
- Client

1 Call  | 2 Calls Serial  | 2 in Bulk  | 100 in Bulk

© Copyright 2013 OSIsoft, LLC.
## Getting Data – New Bulk Calls to PI

<table>
<thead>
<tr>
<th>Capability</th>
<th>PIPoint</th>
<th>PIPointList</th>
<th>AFAttribute / AFData</th>
<th>AFAttributeList / AFListData</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snapshot</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes (AFAttribute.GetValue)</td>
<td>Yes</td>
</tr>
<tr>
<td>Recorded Value</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interpolated Value</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Summary</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Recorded Values</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Interpolated Values</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Plot Values</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Summaries</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Filtered Summaries</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Annotations</td>
<td>Yes</td>
<td>No</td>
<td>Yes (PIPoint DR only)</td>
<td>No</td>
</tr>
<tr>
<td>Update Value</td>
<td>Yes</td>
<td>n/a</td>
<td>Yes</td>
<td>n/a</td>
</tr>
<tr>
<td>Update Values</td>
<td>Yes</td>
<td>Yes via PIServer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Calculated Values</td>
<td>Yes (AFCalculation)</td>
<td>No</td>
<td>Yes (AFCalculation)</td>
<td>No</td>
</tr>
<tr>
<td>Data Pipe</td>
<td>NA</td>
<td>Yes (PIDataPipe)</td>
<td>NA</td>
<td>Yes (AFDataPipe)</td>
</tr>
</tbody>
</table>
New Bulk Call Overview

- New RPC in PI Server 2012
- Allows requests for multiple points to be batched into a single request
- Caller can control how results are paged to the client (by Tag Count, by Event Count)
- Much Faster than serial
- Somewhat faster than making parallel call
New Bulk RPC Example

AFListData.PlotValues

IEEnumerable of same return type as non-bulk call

```
C#
public IEnumerable<AFValues> PlotValues(
    AFTimeRange timeRange,
    int intervals,
    PIPagingConfiguration pagingConfig
)
```

Specify how data is paged back from client
Bulk RPC Example - Paging

PIPagingConfiguration parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BulkPayloadPercentThreshold</td>
<td>This value represents a percentage of the entire RPC that must be completed to cause a page to be proactively returned to prevent the operation timeout from expiring.</td>
</tr>
<tr>
<td>KeepAliveTimeout</td>
<td>This value is the maximum amount of time allowed to elapse between calls to get the next page of results.</td>
</tr>
<tr>
<td>OperationTimeoutOverride</td>
<td>This value overrides the operation timeout set on the PIServer for the duration of the data access call. The operation timeout is the maximum amount of time that can elapse on the PIServer while fetching each page.</td>
</tr>
<tr>
<td>PageSize</td>
<td>The size of the pages that will be returned depending on the PageType.</td>
</tr>
<tr>
<td>PageType</td>
<td>The PageType is used to determine how partial results from list data access calls should be grouped while being returned to the client. Choosing the appropriate page type will improve performance.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Member name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TagCount</td>
<td>0</td>
<td>Results are paged from the PIServer by tag count, with a maximum tags per page specified in the PageSize property.</td>
</tr>
<tr>
<td>EventCount</td>
<td>2</td>
<td>Results are paged from the PIServer by event count. A new page of data is retrieved when the number of events exceeds the PageSize property. All events for a single tag are retrieved in one page.</td>
</tr>
</tbody>
</table>
Bulk RPC Example - Paging

```csharp
private double GetDataInBulk(AFAttributeList attributes, AFTimeRange timeRange)
{
    // Defined paging strategy
    PIPagingConfiguration pagingStrategy = new PIPagingConfiguration(
        PIPageType.TagCount, 1000);

    // make call. Return is an Enumerator
    var results = attributes.Data.Summary(
        timeRange, AFSummaryTypes.Total,
        AFCalculationBasis.EventWeighted, AFTimestampCalculation.Auto,
        pagingStrategy);

    // paging will occur behind the scenes as we enumerate
    double total = 0;
    foreach (var result in results)
    {
        AFValue v = result[AFSummaryTypes.Total];
        if (v.IsGood && v.Value is double)
        {
            total += (double)v.Value;
        }
    }

    return total;
}
```
Bulk Performance

Recorded Values
PI Server 2012
25K points
3 million events retrieved
Bulk Performance

Summaries
PI Server 2012
25K points
1 summary per point
1000 summed events per point
New Bulk Call Behaviors

• Native support on PI Server 2012 and later
• AF SDK will parallelize calls on older Servers
• Non-PI Data Reference Attributes may not use Bulk RPC
• If Filter is not the exact same for all points, call will be done in parallel, not bulk.
• Bulk more predictable than Parallel
• Bulk works better on low-powered clients
Bulk calls to AF Server

• Use Search over Hierarchy
• Load In Bulk
• Load Partial Elements

Caution:
– Recursive hierarchy walking
– For Each on top-level collections
– Complex Configurations (Summary Data References)
– References out-of Element in Data References
– Custom Data References which get own inputs
– Large AF Tables (use Parameterized Tables in AF 2014)
Make Calls In Parallel

• Bulk calls not always possible
  – Too hard to setup in your application
  – Not the same request across points
  – You have a lot of Client side processing

• Utilize idle client Cores

• Example – processing Event Frames
Parallel Example

```csharp
private double GetDataInParallel(AFAtributeList attributes)
{
    double total = 0;
    object monitor = new System.Object();

    // process in parallel. Each summary is a different time range
    // so no bulk call is available
    Parallel.ForEach(attributes, attribute =>
    {
        AFEEventFrame ef = attribute.Element as AFEEventFrame;
        var result = attribute.Data.Summary(
            ef.TimeRange, AFSummaryTypes.Total,
            AFCalculationBasis.EventWeighted, AFTimestampCalculation.Auto);

        AFValue v = result[AFSummaryTypes.Total];
        if (v.IsGood && v.Value is double)
        {
            // need to lock when accessing shared "total"
            lock (monitor)
            {
                total += (double)v.Value;
            }
        }
    });
    return total;
}
```
Parallel Cautions

• Lots of small RPCs not as efficient as Bulk
  o For Server, Client, and Network

• Coding is more complex
  o Locking required

• Mileage varies more
  o Differing capabilities on different computers

• You can overstress a system
  o use Tasks and Parallel constructs to reduce stress
Stream Processing
Watching for data flowing into PI and processing only changes

• Support for new Abacus Scheduling Engine

• New AF Classes
  AFDataPipe
  AFDataCache
Data Pipe


AF Data Pipe maintains client side cache of inputs for some calculation based Data References. PI Data Pipe has no client cache.
Data Pipe

- PIDataPipe released in AF 2012 (2.5) for PIPoints Snapshot, Archive. Adding Combined in PI 2014
- AFDataPipe new in AF 2014 (2.6) for all Attributes
- New IObservable Pattern
- It is the Most Efficient way to get real-time data
  - Done in bulk
  - Server only sends Changes of snapshot
  - Server buffers events between client fetch
- Run multiple pipes in parallel
AFDataPipe Example

Signup

```csharp
AFDataPipe _dataPipe;
System.Timers.Timer _timer;
private void Signup(AFAttributeList attributes)
{
    _dataPipe = new AFDataPipe();
    var errors = _dataPipe.AddSignups(attributes);
    if (errors.HasErrors)
    {
        ProcessErrors(errors);
    }
    else
    {
        _timer = new System.Timers.Timer();
        _timer.Interval = 1000;
        _timer.Elapsed += TimerElapsed;
        _timer.Start();
    }
}
```

Process Events

```csharp
private void TimerElapsed(object sender, EventArgs e)
{
    bool moreEvents = true;
    while (moreEvents)
    {
        var results = _dataPipe.GetUpdateEvents(100, out moreEvents);
        foreach (AFDataPipeEvent evt in results.Results)
        {
            ProcessEvent(evt);
        }
        _timer.Start();  // restart the timer.
    }
```
AFDataPipe Observer Pattern

Signup

```csharp
EventObserver _observer;
private void SignupObserver(AFAttributeList attributes)
{
    _dataPipe = new AFDataPipe();
    var errors = _dataPipe.AddSignups(attributes);

    _observer = new EventObserver();
    _dataPipe.Subscribe(_observer);
    _timer = new System.Timers.Timer();
    _timer.Interval = 1000;
    _timer.Elapsed += delegate(object o, ElapsedEventArgs e)
    {
        bool moreEvents = true;
        while (moreEvents)
        {
            _dataPipe.GetObserverEvents(100, out moreEvents);
        }
    }
    _timer.AutoReset = true;
    _timer.Start();
}
```

Process Events

```csharp
public class EventObserver : IObserver<AFDataPipeEvent>
{
    private int _count = 0;
    private bool _isComplete = false;

    #region IObserver<AFDataPipeEvent> Members
    public void OnNext(AFDataPipeEvent evt)
    {
        ProcessEvent(evt);
    }

    public void OnCompleted()
    {
        _isComplete = true;
    }

    public void OnError(Exception error)
    {
    }
    #endregion
}
```

Observer Pattern removes list creation, improving throughput
Data Pipe – Events per Second/Thread

Events/Sec

AF 2.5 | AF 2.6 | AF Observer

Elapsed Time

CPU Time

© Copyright 2013 OSIsoft, LLC.
AFDataCache utilizes Data Pipe mechanism for feeding the cache.

AFDataCache maintains cached “AFData” for each attribute. It feeds the cache via UpdateData.
AFDataCache

• Holds Data Client side
• Data can be used multiple times
• Useful when client may not be able to bulk
• Good for Time Synchronizing Data
• When Bulk, Parallel, and Stream are not enough
# AFDataCache

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheTimeSpan</td>
<td>The minimum amount of time series data to be kept for each AFAtribute in the cache, as measured by time span.</td>
</tr>
<tr>
<td>MaxCacheEventsPerAttribute</td>
<td>The maximum number of cache events to be kept for each AFAtribute in the cache.</td>
</tr>
<tr>
<td>MinCacheEventsPerAttribute</td>
<td>The minimum number of cache events to be kept for each AFAtribute in the cache.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds a list of AFAtribute objects to be managed by AFDataCache. The AFAtributes are reference counted. Adding AFAtribute that is already in the AFDataCache will increment the reference count and not generate error. Adding the same AFAtribute multiple times will require the same number of Remove call to actually remove the AFAtribute from the AFDataCache.</td>
</tr>
<tr>
<td>TryGetItem</td>
<td>Returns the cache enabled AFDATA object for a given AFAtribute.</td>
</tr>
<tr>
<td>UpdateData()</td>
<td>Update the time series cache for the attributes managed by by this AFDataCache.</td>
</tr>
</tbody>
</table>
AFDataCache Example

Setup

```csharp
AFDataCache _dataCache;
private void Setup(AFAttributeList attributes)
{
    _dataCache = new AFDataCache(attributes.Count);
    _dataCache.MaxCacheEventsPerAttribute = 10;
    _dataCache.MinCacheEventsPerAttribute = 2;
    _dataCache.CacheTimeSpan = TimeSpan.FromMinutes(2);
    _dataCache.Add(attributes);

    _timer = new System.Timers.Timer();
    _timer.Interval = 1000;
    _timer.Elapsed += delegate(object o, ElapsedEventArgs e)
    {
        _dataCache.UpdateData();
    }
    _timer.AutoReset = true;
    _timer.Start();
}
```

Use

```csharp
private AFValue UseCache(AFAttribute attribute, AFTime time)
{
    AFData afdata;
    // retrieve the cache aware AFData object
    if (!_dataCache.TryGetItem(attribute, out afdata))
    {
        afdata = attribute.Data; // no cache.
    }

    return afdata.InterpolatedValue(time, null);
}
```
AFDataCache Performance

InterpolatedValue
10,000 Attributes
AFDataCache Cautions

• Potential for Stale Data
• High Memory Usage
• Not all AFData calls are cache-aware
• Small practical time range
Pre-Calculate

- Sometimes, there is just too much data to calculate on demand.
- Use Abacus to pre-calculate instead of Data References
- Resultant Data queries can be orders of magnitude faster
Pre-Calculate - Tradeoffs

• Late Arriving Data
• Back-Calculation
• Fidelity of Data
• Additional Tags
Pre-Calculate: Point Path Resolution

PI System Explorer shows simplified name:
\myPIServer1\Tank101.Temperature

Actual stored string may differ:
• Calculated from the template:
  \%@myServer\%\Element\%\Attribute\
• Unresolved:
  \myPIServer1\Tank101.Temperature
• Resolved:
  \myPIServer1\?fa2958bc-b5ab-4686-a5f2-16155eb0ab71\Tank101.Temperature?10891

Server ID
Point ID
Point Path Resolution- How do you Know?

Icon (AF 2.6 / 2014)
Indicates point could be auto created or resolved

Tooltip (2.6 / 2014)

AFExport (all AF Versions)

C:\Program Files\PIPC\AF\AFExport "\\ MyDatabase \ MyElement | Temperature" | findstr ConfigString <ConfigString>\\myPIServer 7bb700a7-2df3-433a-943b-9f0d167fa2f3\CDT158?3</ConfigString>
Point Path Resolution

Resolve in PSE

New Command Line Tool

C:\Program Files\PIPC\AF\AFUpdatePluginConfiguration /Resolve "\:\MyDatabase"

Time = 26.6 s
Diagnosing Performance Tools

- **RPC Metrics**
  - RPC metrics to af server, client measurements

- **RPC Metrics**
  - RPC metrics to af server, server measurements

- **SQL Tracing**
  - Enable and report through AFDiag utility

- **AFGetTrace**
  - Client call level trace detail

- **PI System Management**
  - RPC metrics to PI server, server measurements
Summary
Strategies for Optimizing SDK Applications

• Bulk
• Parallel
• Stream (Data Pipe)
• Cache on Client
• Pre-Calculate
• Diagnose Performance Bottlenecks

Remember – at large scale – everything matters
Optimizing Server Side

“You can’t have more than there is”
Paul Combellick.
Optimizing your AF Server
PI AF Server Topics

• Deployment options for PI AF 2.5 (PI AF 2012)
• New or Improved in PI AF 2.6 (PI AF 2014)
Deployment

- Recovery Model
- SQL Server Editions / Versions
- Database File Sizing
- Memory Sizing
- Disk Configuration
Deployment SQL Server Recovery Models

- **Simple** – default for a new AF installation
  - Transaction log does not grow.
  - Best for test systems or any system that you can rebuild the data.
    - `ALTER DATABASE [PIFD] SET RECOVERY SIMPLE;`

- **Full**
  - Transaction log grows until log backup.
  - Best practice for production systems.
  - Can recover post-backup data changes after data disk failure.
    - `ALTER DATABASE [PIFD] SET RECOVERY FULL;`

SQL Server Recovery Models
Best Practice

– Use Simple Recovery for Dev / Test systems.

– Use Full Recovery for Production systems.
Deployment
SQL Server Versions / Editions

Versions

- PI AF 2.5 supports SQL Server 2005 – 2012
- PI AF 2.6 supports SQL Server 2008 – 2012 (Testing with SQL Server 2014 CTP2 and SQL Azure)

Editions

- Express
  - Severely limited: 1GB RAM, 1 CPU, data + log size limited to 10GB, No job scheduler (SQL Agent),
  - Effectively limited to a few tens of thousands of AF Objects

- Standard
  - Max 16 cores, 64GB RAM
  - Clustering, mirroring, AF Collective primary

- Enterprise
  - RAM & Max Cores limited by Windows
  - Clustering, mirroring, AF Collective primary, Always-On
  - Change Data Capture – PI AF2.6 Audit trail

- Developer – included in MSDN license

SELECT @@SERVERNAME, @@VERSION ;
SQL Server Versions / Editions
Best Practice

• Small Systems ( < 1.5 GB data, a few users)
  – SQL Express

• Require high availability, compressed backups, large amount of RAM or CPU, PI AF 2014 Audit trail:
  – SQL Server 2012 Enterprise Edition
Deployment
PIFD Database File Sizing

• Auto growth of database files adversely affects write performance except on SSD.
Deployment
PIFD Database File Sizing

• Auto growth of database files adversely affects write performance.

• PI AF 2.5: Files start at 1MB and grow by 1MB
  • A deployed system can be modified using SSMS

• PI AF 2.6: Files start at 1MB and grow by 100MB
Deployment
PIFD Database File Sizing

Time (seconds) to insert 50K AFEElements

PI AF 2.5
SQL Express 2012

Option A: 1MB, GROW 1MB
Option B: 1MB, GROW 100MB
Option C: 1GB, NO GROWTH
PIFD Database File Sizing
Best Practice

• Pre-allocate file size, except on SSD.

• Disable AUTO_SHRINK.

• Don’t shrink database files.
Deployment
Memory Sizing

• Quantity of RAM allocated to SQL Server can affect read and write performance.

• Generally, want RAM allocated to SQL Server Instance to exceed 60% of the size of the database(s).
Deployment
Disk Configuration

Time (seconds) to insert 50K AFEElements

PI AF 2.5
SQL Express 2012
Test Platform: Dell Precision T7600

- 32 GB RAM
- 24 logical CPU
- 256GB SSD – 40K IOPS
- 1TB drive
- Windows 8
- SQL 2012 Dev
- SQL Express 2012
- 15 SAS Drives 146 GB
- 4.5K IOPS
Disk Configuration
Best Practice

• Choose drive configuration with sufficient IOPs, bandwidth, storage capacity, and availability.
# Deployment High Availability Options

<table>
<thead>
<tr>
<th></th>
<th>SQL Express</th>
<th>SQL Cluster</th>
<th>SQL Mirror</th>
<th>AF Collective</th>
<th>SQL Always-On</th>
</tr>
</thead>
<tbody>
<tr>
<td>HA Reads</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HA Writes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Quick Failover</td>
<td>NA</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of Data Copies</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2+</td>
<td>2-4</td>
</tr>
</tbody>
</table>
High Availability Options
Best Practice

• 2 or more AF Servers, load balanced

• SQL Server 2012 Always-On
  – New in SQL 2012, Mirroring ++
  – PI AF 2.5 compatibility white paper available.
New or Improved in PI AF 2.6

• Finding Elements By Path
• Insert Performance
• Hierarchical Template Performance
• Maintenance Job
• AF Sql Trace
Improved in PI AF 2.6

AFEElement.FindElementsByPath Method

AFEElement Class See Also Send Feedback

Retrieves the AFEElement objects identified by the path strings.

<table>
<thead>
<tr>
<th>Overload List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>———</td>
</tr>
<tr>
<td>FindElementsByPath(IEnumerable&lt;string&gt;, AFOBJECT)</td>
</tr>
<tr>
<td>FindElementsByPath(IEnumerable&lt;string&gt;, AFOBJECT, IDictionary&lt;string, string&gt;)</td>
</tr>
</tbody>
</table>

\AFServerName\AFDatabaseName\Meters\AcmeHeadEnd00001006\Task1
PI AF 2.5 Searching Metrics

Search Time, Small Result Sets

Database size, Elements

Milliseconds

Path

Name a*
AFEElement.FindElementsByPath ( )

- FindElementsByPath() is used by several client tools.
- Too slow, especially > 10K elements.
- If more than a few concurrent requests, response slows to a crawl, SQL Server CPU -> 100%.

- Can we improve performance?
- Can we improve concurrency?
- What are the tradeoffs?
AFEElement.FindElementsByPath()
Performance - AF2.5 vs AF2.6

<table>
<thead>
<tr>
<th>AFEElement Count</th>
<th>AFElement Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AF 2.5</td>
</tr>
<tr>
<td>2,000</td>
<td>AF 2.6</td>
</tr>
<tr>
<td>4,000</td>
<td></td>
</tr>
<tr>
<td>6,000</td>
<td></td>
</tr>
<tr>
<td>8,000</td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td></td>
</tr>
<tr>
<td>12,000</td>
<td></td>
</tr>
<tr>
<td>14,000</td>
<td></td>
</tr>
<tr>
<td>16,000</td>
<td></td>
</tr>
<tr>
<td>18,000</td>
<td></td>
</tr>
<tr>
<td>1,000,000</td>
<td></td>
</tr>
</tbody>
</table>
### AFEElement.FindElementsByPath()

**Performance - AF2.5 vs AF2.6**

<table>
<thead>
<tr>
<th>AFEElement Count</th>
<th>AFElement Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AF 2.5</td>
</tr>
<tr>
<td>0</td>
<td>1,000</td>
</tr>
<tr>
<td>2,000</td>
<td></td>
</tr>
<tr>
<td>4,000</td>
<td></td>
</tr>
<tr>
<td>6,000</td>
<td></td>
</tr>
<tr>
<td>8,000</td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td></td>
</tr>
<tr>
<td>12,000</td>
<td>16,000</td>
</tr>
<tr>
<td>14,000</td>
<td>18,000</td>
</tr>
<tr>
<td>16,000</td>
<td></td>
</tr>
<tr>
<td>18,000</td>
<td></td>
</tr>
</tbody>
</table>

**AF 2.5 > 10,000 X**

**AF 2.6**

**performance improvement**
PI AF 2.6 FindElements...( )
Concurrent Requests

Number of Threads
SQL Express 2012
50K Elements

elements per minute

FindElementsByPath(1)
FindElements(Name)
FindElements(Advanced)
PI AF 2.6 FindElements…( )

Concurrent Requests

Number of Threads

SQL Express 2012

50K Elements

elements per minute

FindElementsByPath(1)

FindElements(Name)

FindElements(Advanced)

FindElementsByPath(100)
PI AF 2.6 FindElementsByPath ( )
Concurrent Requests – Batch Size

Number of Threads

SQL Server 2012 Dev Edition
32GB RAM
6 million Elements
Improved in PI AF 2.6
Insert Performance

AFEElement Insert Time relative to PI AF2.6

Single Drive
SQL 2012 Dev
Improved in PI AF 2.6
Insert Performance

Insert Time relative to AF2.6

SSD SQL 2012 Dev
Improved in PI AF 2.6
Hierarchical Template Performance

FindElementAttributes( ): Elapsed Time relative to AF 2.6
Searching Guidelines

• **Best:**
  – Path Searches
  – 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)
  – Attribute Searches

• **Caution:**
  – Searches with large results sets (Limited by Bandwidth)
  – Complex Criteria (more joins = slower performance)
New for PI AF 2.6 Maintenance Job

- Re-Indexing
- Statistics Maintenance
- Rebuild Path Cache
- Delete Orphaned Elements
- Delete Orphaned Files, AFDatabases
New for PI AF 2.6
Monitoring Performance: AF SQL Trace

- Enable: `afdiag /ST+`
- Disable: `afdiag /ST-`
- Delete: `afdiag /ST0`
- Summary: `afdiag /STP`
Key Performance Points – PI AF Server

• Add RAM to SQL Server.
• Use drives with adequate IO / storage capacity.
• Pre-allocate database files.
• Weekly database Maintenance.
• SQL Server Always-On (Mirroring ++).
Please don’t forget to…

Complete the online survey for this session

eventmobi.com/vcampus13

Share with your friends

#VCL13
THANK YOU