Efficient and scalable cross-matching of (very) large catalogues

François-Xavier Pineau¹, Thomas Boch¹ and Sebastien Derrière¹

¹CDS, Observatoire Astronomique de Strasbourg

ADASS Boston, 08 November 2010
Context

CDS cross-match service (in development)

- Based on UWS (job submission)
- Catalogues:
  - Simbad
  - Vizier
  - XML

- Algorithms:

- Particularity: deal with (very) large catalogues
Context

CDS cross-match service (in development)

- Based on UWS (job submission)
- Catalogues:
  - Simbad
  - Vizier
- Algorithms:
- Particularity: deal with (very) large catalogues
Context

CDS cross-match service (in development)

- Based on UWS (job submission)
- Catalogues:
  - SimBAD
  - VizieR
- Algorithms:
  - Data points
- Particularity: deal with (very) large catalogues
Context

CDS cross-match service (in development)

- Based on UWS (job submission)
- Catalogues:
  - Simbad
  - VizieR
- Algorithms:
- Particularity: deal with (very) large catalogues
Dealing with (very) large catalogues

Example

- **2MASS**
  - ~470x10^6 sources
  - minimal data ~ 15 GB
    - identifier (integer 4 Bytes)
    - positions (double 8 B+8 B)
    - errors (float 4 B+4 B+4 B)

- **USNO-B1**
  - ~10^9 sources
  - minimal data ~ 28 GB
    - identifier (integer 4 B)
    - positions (double 8 B+8 B)
    - errors (float 4 B+4 B)

- **LSST projection at 5 years:**
  - V>26, ~3x10^9 unique sources
  - minimal data ~ 96 GB

Problems

- Data size
  - do not fit into memory

- Performance issues
  - data loading
  - looking for candidates

Solutions

- Scalability: Healpix partitioning
- Efficiency:
  - special indexed binary file
  - kd-tree (cone search queries)
  - multithreading
  - parallel processing
Dealing with (very) large catalogues

Example

- **2MASS**
  - \( \sim 470 \times 10^6 \) sources
  - minimal data \( \sim 15 \) GB
    - identifier (integer 4 Bytes)
    - positions (double 8 B + 8 B)
    - errors (float 4 B + 4 B + 4 B)

- **USNO-B1**
  - \( \sim 10^9 \) sources
  - minimal data \( \sim 28 \) GB
    - identifier (integer 4 B)
    - positions (double 8 B + 8 B)
    - errors (float 4 B + 4 B)

- **LSST projection at 5 years:**
  - \( V > 26, \sim 3 \times 10^9 \) unique sources
  - minimal data \( \sim 96 \) GB

Problems

- **Data size**
  - do not fit into memory

- **Performance issues**
  - data loading
  - looking for candidates

Solutions

- **Scalability:** Healpix partitioning
- **Efficiency:**
  - special indexed binary file
  - \( kd \)-tree (cone search queries)
  - multithreading
  - parallel processing

François-Xavier Pineau (CDS)
Dealing with (very) large catalogues

Example

- 2MASS
  - ~ 470\times 10^6 sources
  - minimal data ~ 15 GB
    - identifier (integer 4 Bytes)
    - positions (double 8 B + 8 B)
    - errors (float 4 B + 4 B + 4 B)

- USNO-B1
  - ~ 10^9 sources
  - minimal data ~ 28 GB
    - identifier (integer 4 B)
    - positions (double 8 B + 8 B)
    - errors (float 4 B + 4 B)

- LSST projection at 5 years:
  - V > 26, ~ 3 \times 10^9 **unique** sources
  - minimal data ~ 96 GB

Problems

- Data size
  - do not fit into memory

- Performance issues
  - data loading
  - looking for candidates

Solutions

- Scalability: Healpix partitioning
- Efficiency:
  - special indexed binary file
  - \textit{kd}-tree (cone search queries)
  - multithreading
  - parallel processing
Healpix

- Hierarchical sky pixelisation
  - level 0 \(\mapsto\) 12 pixels
  - level 1 \(\mapsto\) 12x4 pixels
  - ... 
  - level \(n\) \(\mapsto\) 12x\(2^n\)
- Pixels of equal area
- Developed at NASA: healpix.jpl.nasa.gov
- Available in
  - C, C++
  - Fortran
  - IDL
  - Java
  - ...?
Scalable cross-match

- Independent pixels cross-match
  - but border effects
- Cat. B pixel sources put in a kd-tree
- Optimal partitioning level
  - available memory
  - minimisation of:
    \[
    \sum_{i=0}^{nPixels} N_{A_i} \log(1 + N_{B_i} + N_{B_i}^b)
    \]
  - I/O cost

Level 0

Level 1
Scalable cross-match

- Independent pixels cross-match
  - but border effects
- Cat. B pixel sources put in a kd-tree
- Optimal partitioning level
  - available memory
  - minimisation of:
    \[
    \sum_{i=0}^{n\text{Pixels}} N_{A_i} \log(1 + N_{B_i} + N_{B_i}^b)
    \]
  - I/O cost

Level 0

Level 1

A

B
Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow “on the fly” correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
- Framework:
  - based on UWS (few machines)
  - Hadoop (large grid)
- “On the fly” correlation possible

\(^a\)Universal Worker Service (IVOA)
Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow “on the fly” correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
- Framework:
  - based on UWS\(^a\) (few machines)
  - Hadoop (large grid)
- “On the fly” correlation possible

\(^a\)Universal Worker Service (IVOA)
Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow “on the fly” correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
- Framework:
  - based on UWS\(^a\) (few machines)
  - Hadoop (large grid)
- “On the fly” correlation possible

\(^a\)Universal Worker Service (IVOA)
Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow “on the fly” correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
- Framework:
  - based on UWS\(^a\) (few machines)
  - Hadoop (large grid)
- “On the fly” correlation possible

\(^a\)Universal Worker Service (IVOA)
Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow “on the fly” correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
- Framework:
  - based on UWS\(^a\) (few machines)
  - Hadoop (large grid)
- “On the fly” correlation possible

\(^a\)Universal Worker Service (IVOA)
Loading data: indexed binary files

Binary data file

- Organized by blocks:
  - positions
  - position errors
  - identifiers
  - ...

- Sources ordered by healpix pixel index
Loading data: indexed binary files

**Index files**
- One by healpix level
- For each pixel
  - offset
  - nSources

**Binary data file**
- Organized by blocks:
  - positions
  - position errors
  - identifiers
  - ...
- Sources ordered by healpix pixel index

<table>
<thead>
<tr>
<th>level 0 index file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idx</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

François-Xavier Pineau (CDS)
Loading data: indexed binary files

Index files
- One by healpix level
- For each pixel
  - offset
  - nSources

Binary data file
- Organized by blocks:
  - positions
  - position errors
  - identifiers
  - ...
- Sources ordered by healpix pixel index

Francois-Xavier Pineau (CDS)
X-Match of large catalogues
08/11/2010 7 / 16
What is a $kd$-Tree?

- A space-partitioning data structure
- Allows for fast $k$-nearest neighbour/cone search queries
  - nearest neighbour query in $O(\log(n))$

Problem

- Naive implementation can be memory consuming
- We want a memory efficient $kd$-tree (capacity > 1 billion sources)

Solution

- To use a single array (sorted using a $kd$-tree scheme)
What is a $kd$-Tree?

- A space-partitioning data structure
- Allows for fast $k$-nearest neighbour/cone search queries
  - nearest neighbour query in $O(\log(n))$

Problem

- Naive implementation can be memory consuming
- We want a memory efficient $kd$-tree (capacity $> 1$ billion sources)

Solution

- To use a single array (sorted using a $kd$-tree scheme)
What is a *kd*-Tree?

- A space-partitioning data structure
- Allows for fast *k*-nearest neighbour/cone search queries
  - nearest neighbour query in \( O(\log(n)) \)

Problem

- Naive implementation can be memory consuming
- We want a memory efficient *kd*-tree (capacity > 1 billion sources)

Solution

- To use a single array (sorted using a *kd*-tree scheme)
A kd-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate
A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate

\[
\begin{align*}
\alpha \leq \alpha_{S_3} & \quad \delta \leq \delta_{S_{10}} \\
\alpha_{S_3} & \quad \delta_{S_{10}} \leq \delta \\
\alpha_{S_3} \leq \alpha & \quad \delta_{S_8} \leq \delta \\
\alpha \leq \alpha_{S_{11}} \leq \alpha & \quad \alpha \leq \alpha_{S_6} \leq \alpha \\
\alpha \leq \alpha_{S_4} \leq \alpha & \quad \alpha \leq \alpha_{S_2} \leq \alpha \\
\end{align*}
\]
A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate

Creation speed up by using multi-threading
A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate

Creation speed up by using multi-threading
A *kd*-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate

Creation speed up by using multi-threading
A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate

**Thread 1**

Creation speed up by using multi-threading
A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate

```
α
S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15
δ
```

Creation speed up by using multi-threading
A \( kd \)-tree can be a simple sorted array of sources

Algorithm: \textit{quicksort} alternating the sorted coordinate

Thread 1

Thread 2

Thread 3

Thread 4

Creation speed up by using multi-threading
Modified $kd$-tree and multithreading

Modified $kd$-tree

- Classical $kd$-tree adapted for euclidian spaces
- Solution 1: (rejected)
  - cartesian coordinates $(x, y, z)$
    - $\leadsto$ time consuming (conversion)
    - $\leadsto$ memory consuming (+50%)
- Solution 2: (approved)
  - spherical coordinates $(\alpha, \delta)$
  - classical creation algorithm
  - modified query algorithm
    - angular distances (Haversine formula)
    - modified circle/rectangle intersection to enter a sub-tree

Multithreading

- Single $k$NN or cone search query not multithread
- Pool of threads executing multiple queries simultaneously
Modified \( kd \)-tree and multithreading

Modified \( kd \)-tree

- Classical \( kd \)-tree adapted for euclidian spaces
- **Solution 1:** (rejected)
  - cartesian coordinates \((x, y, z)\)
    - \(\mapsto\) time consuming (conversion)
    - \(\mapsto\) memory consuming (+50%)
- **Solution 2:** (approved)
  - spherical coordinates \((\alpha, \delta)\)
  - classical creation algorithm
  - modified query algorithm
    - angular distances (Haversine formula)
    - modified circle/rectangle intersection to enter a sub-tree

Multithreading

- Single \( kNN \) or cone search query not multithread
- Pool of threads executing multiple queries simultaneously
Modified kd-tree and multithreading

Modified kd-tree

- Classical kd-tree adapted for euclidian spaces
- Solution 1: (rejected)
  - cartesian coordinates \((x, y, z)\)
    - \(\leadsto\) time consuming (conversion)
    - \(\leadsto\) memory consuming (+50%)
- Solution 2: (approved)
  - spherical coordinates \((\alpha, \delta)\)
  - classical creation algorithm
  - modified query algorithm
    - angular distances (Haversine formula)
    - modified circle/rectangle intersection to enter a sub-tree

Multithreading

- Single kNN or cone search query not multithread
- Pool of threads executing multiple queries simultaneously
Test Machine

- Dell machine 2 600€ ($\sim$3 600):
  - 24 GB of **1333 MHz** memory
  - 2x Quad Core 2.27 GHz (Xeon)
  - **16 threads** (Hyper-Threading)
  - High speed HDD (10 000 rpm)
Test results

Full catalogue cross-correlation

SDSS DR7 (∼357 000 000 sources)  
2MASS (∼470 000 000 sources)

- Simple cross-match: ∼9 min
  - radius of 5″
  - Healpix level 3 (∼7.3°)
  - Level 9 borders (∼7′)
  - ∼49 209 000 associations

- With elliptical errors: ∼10 min
  - distance of 3.44σ
  - distance max of 5″
  - Healpix level 3
  - ∼37 507 000 associations
Test results

Full catalogue cross-correlation

SDSS DR7 (∼357 000 000 sources)

Simple cross-match: ∼9 min
- radius of 5″
- Healpix level 3 (∼7.3°)
- Level 9 borders (∼7’)
- ∼49 209 000 associations

With elliptical errors: ∼10 min
- distance of 3.44σ
- distance max of 5″
- Healpix level 3
- ∼37 507 000 associations

Francois-Xavier Pineau (CDS)
Test results

Full catalogue cross-correlation

SDSS DR7 (∼357 000 000 sources) 2MASS (∼470 000 000 sources)

- Simple cross-match: ∼9 min
  - radius of 5″
  - Healpix level 3 (∼7.3°)
  - Level 9 borders (∼7′)
  - ∼49 209 000 associations

- With elliptical errors: ∼10 min
  - distance of 3.44σ
  - distance max of 5″
  - Healpix level 3
  - ∼37 507 000 associations
Lessons learned

Hardware

For our application:
- RAM frequency *does* matter (lots of memory access)
- Hyper-Threading *does* matter (on 8 cores, 16 threads ∼ 2x faster than 8 threads)

Software: don’t have *a priori*

- Efficient full Java code
- Efficient modified kd-trees (in our case)

Service

- Existing and future (very) large catalogues can be processed
- Bottleneck is data transfer (without surprise)
  - service collocated with data
Test results

Full all-sky catalogues cross-correlation

2MASS (∼470 000 000 sources)  
USNO-B1 (∼1 046 000 000 sources)

- Simple cross-match: ~30 min
  - radius of 5”
  - Healpix level 3
  - Level 9 borders
  - ~583 300 000 associations
Basic likelihood ratio (LR)

Ratio between:

- **Rayleigh distribution**
  \[
  LR = \frac{r \exp \left( -\frac{1}{2} r^2 \right)}{2 \lambda r} = \frac{\exp \left( -\frac{1}{2} r^2 \right)}{2 \lambda}
  \]

- **Poissonian distribution**
  Depends on:
  - \( r = \) normalized distance in \( \sigma \)
  - \( \lambda \propto \) local density of sources

**Test results**

- SDSS7 x 2MASS correlations + LRs
- Local densities estimated by kNN averaging (\( k=100 \))
- \( \sim 15 \text{ min} \)
Basic likelihood ratio (LR)

Ratio between:

- Rayleigh distribution

\[ LR = \frac{r \exp \left( -\frac{1}{2} r^2 \right)}{2\lambda r} = \frac{\exp \left( -\frac{1}{2} r^2 \right)}{2\lambda} \]

- Poissonian distribution

Depends on:

- \( r = \) normalized distance in \( \sigma \)
- \( \lambda \propto \) local density of sources

Test results

- SDSS7 x 2MASS correlations + LRs
- Local densities estimated by kNN averaging (\( k=100 \))
- \( \sim 15\) min
Basic likelihood ratio (LR)

Ratio between:
- Rayleigh distribution
  \[ LR = \frac{r \exp\left(-\frac{1}{2}r^2\right)}{2\lambda r} = \frac{\exp\left(-\frac{1}{2}r^2\right)}{2\lambda} \]
- Poissonian distribution

 Depends on:
- \( r = \) normalized distance in \( \sigma \)
- \( \lambda \propto \) local density of sources

Test results
- SDSS7 x 2MASS correlations + LRs
- Local densities estimated by kNN averaging (\( k=100 \))
- \( \sim 15 \text{min} \)
Going further...

Magnitude-dependent LRs (fast solution)

1. \( \rho(m \pm \Delta m) = \rho \cdot p(m \pm \Delta m) \)
2. kNN averaging
3. log N-log S law
4. SDSS7, level 6 (\( \sim 1^\circ \)), 15 187 non empty histograms computed in 30s.

Probability of identifications (fast solution)

1. Number of spurious match estimates
   - Positional errors sampling for both catalogues
     \[ N_{spur} = \sum_A \sum_B S_{conv}/S_{pixel} \]
2. SDSS7 x 2MASS, level 6, 8min (not yet multithreaded!)
Going further...

Magnitude-dependent LRs (fast solution)

- \( \rho(m \pm \Delta m) = \rho_p(m \pm \Delta m) \)
- \( k\text{NN averaging} \)
- \( \log N \log S \text{ law} \)
- SDSS7, level 6 (~1°), 15,187 non empty histograms computed in 30s.

Probability of identifications (fast solution)

- Number of spurious match estimates
  - Positional errors sampling for both catalogues
  - \( N_{\text{spur}} = \sum_A \sum_B S_{\text{conv}} / S_{\text{pixel}} \)
- SDSS7 x 2MASS, level 6, 8min (not yet multithreaded!)