cross match 软件详细设计说明书

版本：V1.0

更新日期：2013/1/18

作者：徐洋

目录

更新记录

次数	更新内容	更新时间	更新人
1	V1.0 创建文档	2013/1/18	徐洋
2	V1.1		
3			
4			
5			
6			
7			
8			
9			

1. 软件功能

对两个星表 reference 和 sample 进行交叉匹配操作，找出 sample 中与 reference 中相匹配的星（如果两颗星的大圆距离小于 errorRadius，则认为这两颗星匹配），并对匹配成功的星表进行流量判断及流量归一化操作，同时对 sample 中未匹配成功的星的位置进行判断，找出位于 reference 星场之内的星。

2. 软件框架

目前该软件主要包括六个模块：初始化数据库参数，从星表文件中读取数据，交叉匹配，结果分类，流量归一化和结果入库，具体如图 1 所示。

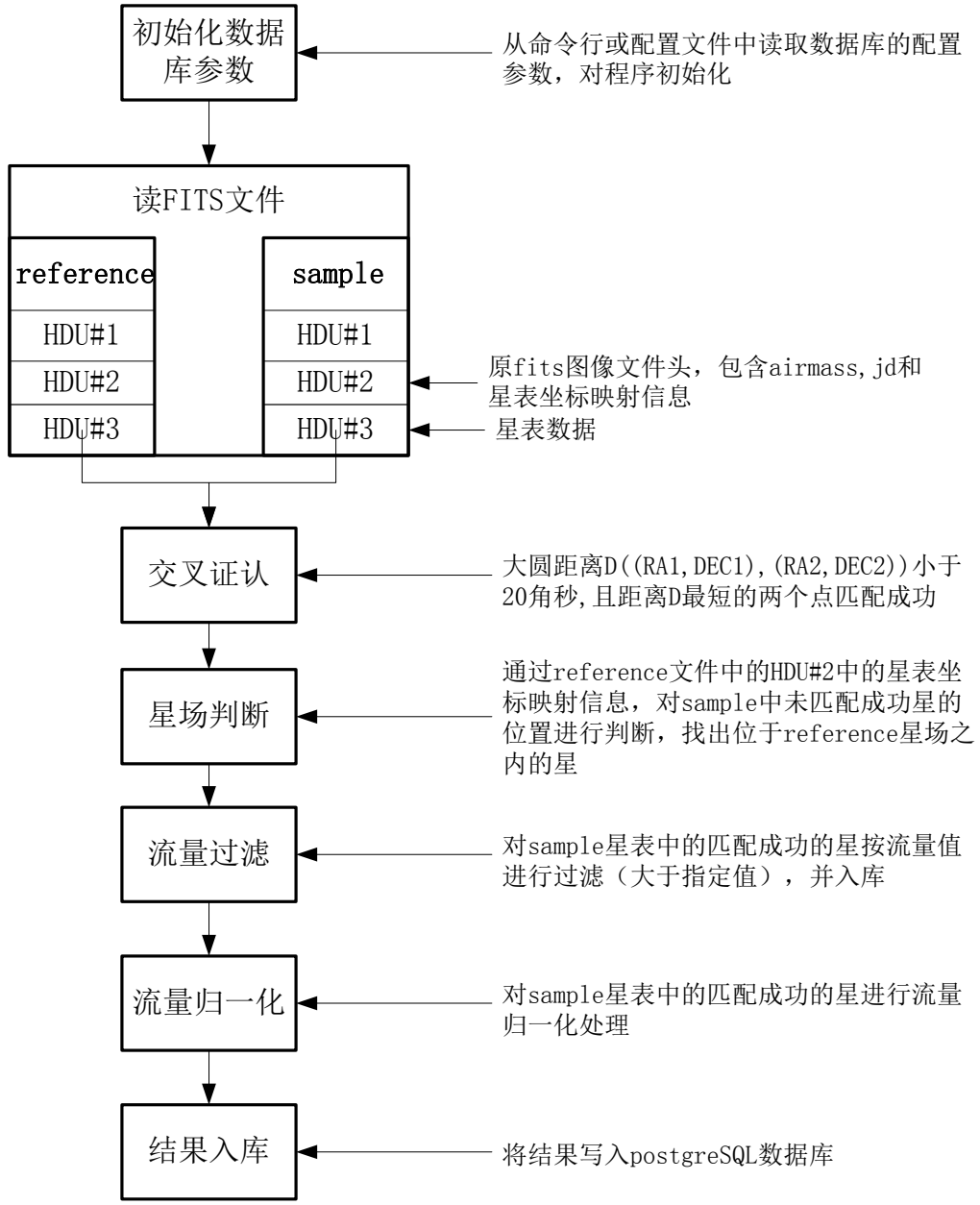

图 1 程序主要功能模块

2.1. 初始化数据库参数

程序中可以分别从命令行和文件对数据库的访问信息进行配置，当命令行和文件同时指定时，优先使用命令行中的信息。数据库的配置信息包括：

主机名称： host=localhost
数据库端口： port=5432
数据库名： dbname=svomdb
数据库用户名：user=wanmeng
数据库密码： password=
Catfile 表： catfile_table=catfile_id
匹配表： match_table=crossmatch_id
未匹配表： ot_table=ot_id
大流量表： ot_flux_table=ot_flux_table

函数列表：

Void getDatabaseInfo(char *configFile) ；

函数功能：从文本文件 configFile 中读取数据库的基本信息

输入：文本文件名 **configFile**

输出：数据库的配置信息

对文本文件数据格式要求：文件中的各个属性必须是“name1=value1,name2=value2…”。

2.2. 读星表文件

星表文件主要以 FITS 格式进行存储，当前项目中的 FITS 文件包含 3 个数据区（HDU）。第一个数据区没有存储数据，第二个数据区中存储原 FITS 图像文件的数据头信息，是一个一行一列的数据块。第三个数据区中存放星表数据，是一个 N 行 M 列的表格，N 行代表有 N 颗星的信息，M 行代表每颗星有 M 个属性。程序主要用到第二个和第三个数据区里的信息。

程序直接使用 cfitsio 库来对 FITS 文件进行读写操作。当用 cfitsio 库读出第二个数据区中的数据后，该数据实际是一个完整的 FITS 数据区头，不能直接进行操作，需要用到 wcs 库，将该数据直接交给 wcs 库进行处理，从中读取 AIRMASS 和 JD，而且后面的结果分类环节将会通过这个数据头里面的星表映射关系对未匹配的结果进行分类。

函数列表：

struct SAMPLE *getSampleFromFits(char *fileName, int *lineNum);

函数功能：从 FITS 文件中读取所有星表数据

输入：FITS 文件名 fileName

输出：星表链表 struct SAMPLE，星的总个数 lineNum

对 FITS 文件星表数据列的格式要求：星表的数据列依次应该按照 id，ra，dec，pixx，pixy，mag，mage，thetaimage，flags，ellipticity，classstar，background，fwhm，vignet 的次序进行排列。

double getFieldFromWCSFloat(char *fileName, int wcsext, char *field);

函数功能：从 FITS 文件的特定数据区中读取指定的属性的值，对应属性为 double 类型

输入：FITS 文件名 fileName，数据区 HDU 编号，属性名称 field

输出：指定属性的值，double 类型

2.3. 交叉证认

2.3.1. 建立索引

对 reference 星表建立分区索引，目前有两种实现方式：

直接经纬度分区索引：按照经度 with=maxRa-minRa 度，纬度 height=maxDec-minDec 度将整个天区分成(with/SIZE)*(height/SIZE)个区域，SIZE 为分区在经纬方向的长度，匹配时直接对各个子区域进行比较，减少比较的次数。如图 2 所示，将 reference 表分为四个区域。

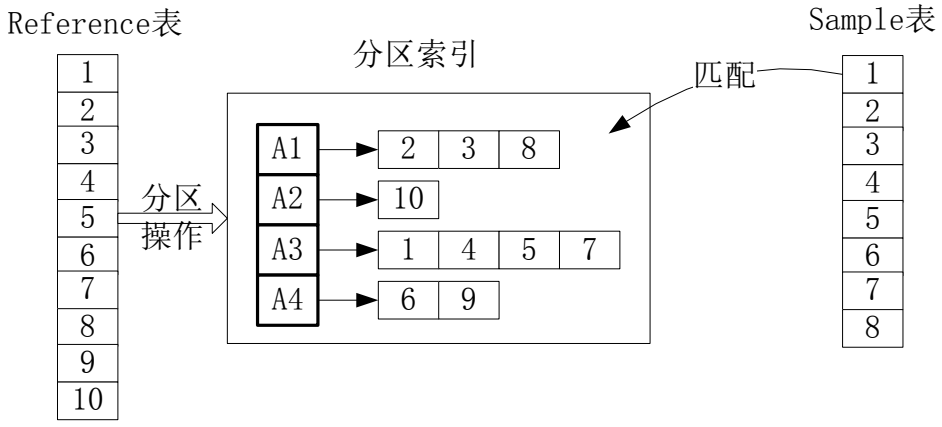

图 2 星表分区及匹配示意图

程序 SIZE 的取值是以索引的内存空间大小为 1MB（左右）为基准的，即：

(with/SIZE)*(height/SIZE)*sizeof(struct AREANODE)=1MB (1)

对 SIZE 向上取整，如果 with 和 height 都很小，而导致 SIZE 的小于 errorRadius，这样程序的效率会比较低。为避免这种情况，在程序中如果 SIZE<5*errorRadius，则令 SIZE=5*errorRadius。当 with=height 时，分区大小为 256*256。

当每个分区中的星的密度比较大时，对每个分区中的星按 ra 或者 dec 进行排序，这样在后面匹配操作时，会进一步减少比较的范围，提高匹配时间。

采用分区算法，就会出现一个分区中的星的匹配星在另一个分区的情况，在直接经纬度分区中可分为四类。如图 3 所示，图中的矩形代表分区，圆形代表一颗星在匹配操作时进行比较操作所覆盖的区域。图中四个圆形分别代表四种情况，其覆盖区域分别为 1 到 4 个。

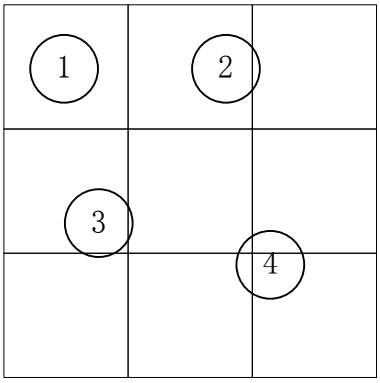

图 3 直接经纬度分区方案中 4 种邻近区域

HTM 分区索引：HTM 分区与直接经纬度分区的过程基本相同，唯一不同的是索引的计算和星的邻近区域判断，HTM 索引计算技术已经非常成熟，其原理请看 Alexander S. Szalay 的文章[6]，程序中直接使用 HTM 库来完成 HTM 索引的计算，使用 Dif 库来计算星的邻近区域。HTM 邻近区域也是四种情况，如图 4 所示，一个星的搜索覆盖区域有 1 个、2 个、3 个和 6 个。

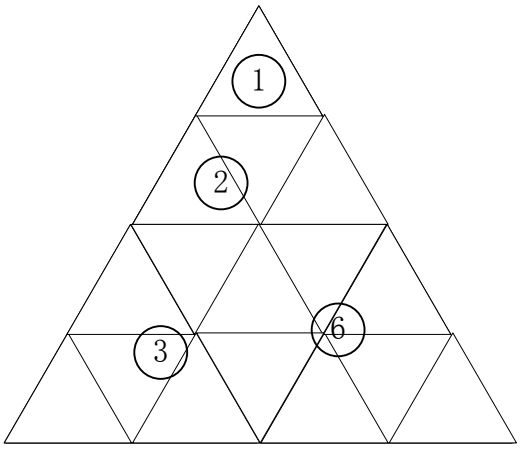

图 4 HTM 分区方案中 4 种邻近区域

根据大圆距离 r 计算同纬度 dec 的经度之差Δ ra：

$$\Delta \mathrm{ra} = \mathrm{acos}\frac{\cos(\mathrm{r}) - \sin^2 \mathrm{dec}}{\cos^2 \mathrm{dec}} \qquad (2)$$

函数列表：

void getAreaBoundary(struct SAMPLE *head);

函数功能：将 reference 星表中 ra 和 dec 的最大值最小值

输入：reference 星表链表头指针 head

输出：ra 和 dec 的最大值最小值（全局变量）

void getZoneLength();

函数功能：根据上面介绍的 SIZE 的取值取值方式，获取分区的大小

输入：ra 和 dec 的最大值最小值

输出：分区的大小

float getAngleFromGreatCircle(double dec, double errorRadius);

函数功能：计算纬度为 dec 的两个点在大圆距离为 errorRadius 时的经度之差。

输入：纬度值 dec，大圆距离 errorRadius

输出：经度之差

void initAreaNode(struct AREANODE *areaTree);

函数功能：初始化分区索引数组

输入：分区索引数组首地址 areaTree

输出：分区索引数组首地址 areaTree

long addDataToTree(struct SAMPLE *head, struct AREANODE *areaTree);

函数功能：将 reference 星表 head 添加到分区索引所对应的各个分区

输入：星表链表头指针 head，分区索引数组 areaTree

输出：分区索引数组 areaTree，添加的星的个数

2.3.2. 交叉匹配

对 reference 和 sample 星表中的星进行交叉匹配操作，对两个星表中的赤经 ra 和赤纬 dec 按大圆距离进行计算，找出距离小于 errorRadius 且数值最小的匹配结果。如图 2 所示，依次循环 sample 中的每颗星，首先计算出当前星所对应的分区索引值，然后将该星依次与每个分区中的每颗星进行比较，找出满足要求的星。

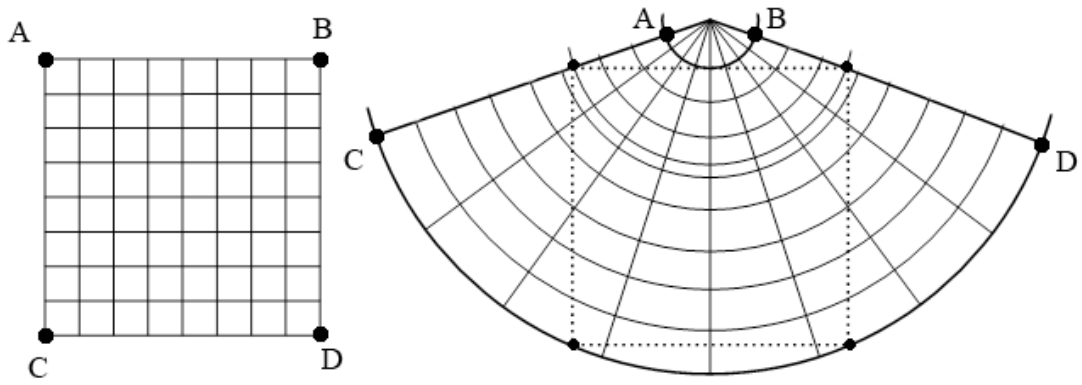

图 5 分区示意图，左为理想分区，右为实际分区

假设原始星表所对应的区域为正方形，如图 5 所示，左边为理想分区，所分的区域面积差距很小，右边为实际分区，最小分区面积与最大分区面积相差几倍。纬度越大的分区边所对应的弧度的长度越短，图 6 为纬度相同的两点的大圆距离为 20 角秒时，其经度之差Δ ra随纬度的变化曲线。可以看出在搜索半径 searchRadius 不变的，随着纬度的增大，经度之差Δ ra 是增大的。图 6 对应公式(2)。

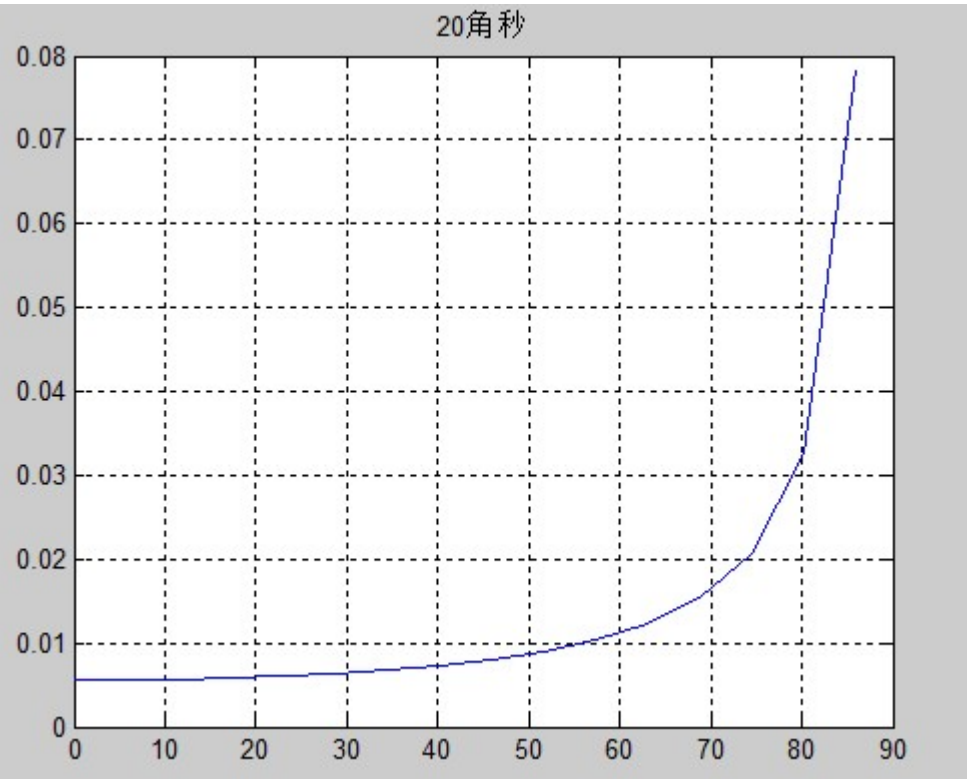

图 6 大圆距离 20 角秒同纬度两点的经度之差Δ ra的变化曲线

计算指定星的搜索分区索引：

1. 比较 down=(dec-searchRadius)和 up=(dec+searchRadius)，得到绝对值最大的一个，赋值给 maxDec；
2. 根据公式(2)计算 maxDec 在 dec+searchRadius 时的经度差Δ ra；
3. 根据Δ ra计算 left=(ra-Δ ra)和 right=(ra+Δ ra)；
4. 这样 up,down,left 和 right 就确定了一个弧形的搜索区域，根据这四个值计算搜索分区索引。

为减少计算量，上面第二步不用每次对每个星都计算经度差Δ ra，程序中采用建立经度差索引，在每次使用时直接通过索引来获得对应的经度差。

大圆距离公式：

$$\arccos(\sin(dec1)*\sin(dec2)+\cos(dec1)*\cos(dec2)*\cos(|ra1-ra2|)) \quad (3)$$

其中 ra1,ra2 为经度，dec1,dec2 为纬度。

交叉匹配伪代码：

```
List crossMatch(List Reference, List Sample)
{
    ReferenceIndexArray = getZoneIndexArray(Reference);
    while each sample in Sample
    {
```

```
        starIndex = getStarIndex(sample);
        while each reference in ReferenceIndexArray(starIndex)
        {
            If (isSampleStar(sample, reference)==TRUE)
                addToMatched(sample);
        }
    }
}
```

函数列表：

void initRaRadiusIndex()；

函数功能：初始化经度距离（同纬度的两点之间距离为 errorRadius 时经度之差）索引。索引长度为：(absDecMax-absDecMin)/searchRadius 向上取整。

输入：纬度绝对值最大值最小值，搜索半径 searchRadius（可以是 errorRadius 的倍数）

输出：索引 raRadiusIndex

void matchPoints(struct AREANODE *areaTree, struct SAMPLE *dataB)；

函数功能：将 sample 星表中的每颗星依次与 reference 分区中对应的分区进行比较，找出匹配的星。

输入：reference 分区索引数组 areaTree，sample 星表链表 dataB

输出：匹配结果 dataB

int getPointSearchBranch(struct SAMPLE *point, long *branch)；

函数功能：获得当前星向邻近的区域

输入：星信息 point

输出：邻近分区数组 branch，邻近分区数

double getGreatCircleDistance(struct SAMPLE *p1, struct SAMPLE *p2)；

函数功能：计算两颗星的大圆距离

输入：两颗星 p1 和 p2

输出：星的大圆距离值

2.4. 星场判断

待匹配的两个星表的星场直接可能只有一部分是相交的，因此 sample 中一部分没匹配成功的点并不是没有对应的匹配点，而是因为它的位置在 reference 星场之外，需要将星场之外的点去掉，找出真正的未匹配成功的点 OT。如图 7 所示，两个星场只有部分重合。

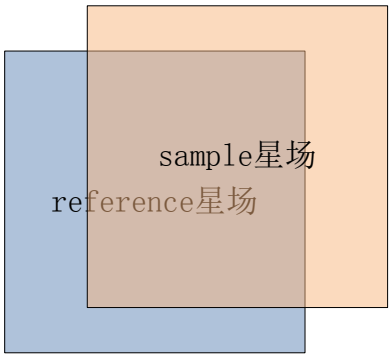

图 7 sample 星场与 reference 星场不完全重合

函数列表：

void wcsJudge(char *fileName, int wcsext, struct SAMPLE *data)；

函数功能：找出 sample 星表中的位于 reference 星场之外的星并设置 isInArea 标记为 1，否则标记为-1

输入：sample 星表 data，reference 的 FITS 文件名 fileName，reference 星表坐标映射 HDU 索引 wcsext

输出：星场判断后的 sample 星表

2.5. 流量归一化

完成 Sample 与 Reference 星表的匹配之后，需要对 Sample 星表的每个匹配上的结果进行流量归一化处理，具体过程如下：

1. 找出匹配结果中 Sample 的 mage 值小于 0.05 的所有点；
2. 对满足条件 1 的所有点计算：$\mathrm{Ratio} = \frac{\mathrm{F_r}}{\mathrm{F_s}} = 10^{-0.4(\mathrm{mag_r} - \mathrm{mag_s})}$，其中$\mathrm{F_r} = 10^{-0.4\mathrm{mag}}$，$\mathrm{F_s} = 10^{-0.4\mathrm{mag}}$；
3. 求出 Ratio 的中值$\mathrm{Ratio_{median}}$
4. 计算归一化因子$\mathrm{magdiff} = -2.5\log_{10}\overline{\mathrm{Ratio}}$；
5. 对 Sample 中每个匹配上的点进行操作：magnorm = mag + magdiff。

函数列表：

double getMedian(double array[], int len)；

函数功能：计算数组的中值

输入：数组 array，数组长度 len

输出：数组中值

float getMagDiff(struct SAMPLE *sample)；

函数功能：计算 sample 星表相对于 reference 星表的归一化因子

输入：sample 星表链表 sample

输出：归一化因子

void fluxNorm(struct SAMPLE *sample, float magDiff)；

函数功能：对 sample 进行归一化操作

输入：sample 星表链表 sample，归一化因子 magDiff

输出：归一化后的 sample 星表

2.6. 流量过滤

完成 Sample 与 Reference 星表的匹配之后，需要对 Sample 星表的每个匹配上的结果按流量进行过滤，具体过程如下：

1. 对 Sample 中所有的匹配成功的星计算：$\mathrm{Ratio} = \frac{\mathrm{F_r}}{\mathrm{F_s}} = 10^{-0.4(\mathrm{mag_r} - \mathrm{mag_s})}$，其中 $\mathrm{F_r} = 10^{-0.4\mathrm{mag}}$，$\mathrm{F_s} = 10^{-0.4\mathrm{mag}}$；
2. 计算 Ratio 的标准差：

$$\mathrm{SD} = \sqrt{\frac{(\mathrm{Ratio_1} - \overline{\mathrm{Ratio}})^2 + (\mathrm{Ratio_2} - \overline{\mathrm{Ratio}})^2 + \cdots + (\mathrm{Ratio_n} - \overline{\mathrm{Ratio}})^2}{\mathrm{n}}}$$

   其中$\overline{\mathrm{Ratio}}$为均值；
3. 对 Sample 中所有匹配成功的星进行过滤，过滤条件：
   如果$|\mathrm{Ratio_i} - \mathrm{Ratio_{median}}| > K * \mathrm{SD}$，则将第 i 颗星的信息写入到 OT_flux_suffix 表中，其中$\mathrm{Ratio_{median}}$ 为上小节中计算的 Ratio 数组的中值，K 为可变参数，由用户输入。

2.7. 数据入库（PostgreSQL）

Sample 星表在与 reference 星表进行匹配及星场判断操作之后，sample 星表中的星被分为 3 类：匹配成功的星 S1，未匹配成功但在 reference 星场之内的星 S2，未匹配成功但在 reference 星场之外的星 S3。S1 会被写入到数据库中的 crossmatch_id 表中，S2 会被写入到 ot_id 表中，S3 不做处理，丢弃。表结构相关信息请参考附录，具体入库流程如下：

1. 判断 reference 星表文件是否在 catfile_id 表中有对应的记录；
2. 如果有，则跳转到第 5 步；
3. 如果没有，则向 catfile_id 表中插入 reference 星表文件信息，对应的 id 为 catid1；
4. 向 crossmatch_id 表中插入 reference 中所有星的信息，其中的属性 catid 值为上面的 catid1；
5. 判断 sample 星表文件是否在 catfile_id 表中有对应的记录；
6. 如果有，则结束；
7. 如果没有，则向 catfile_id 表中插入 sample 星表文件信息，对应的 id 为 catid2；
8. 向 crossmatch_id 表中插入 sample 星表中 S1 的信息，其中的属性 catid 值为上面的 catid2；
9. 向 ot_id 表中插入 sample 星表中 S2 的信息，其中的属性 catid 值为上面的 catid2；

10. 结束

函数列表：

void writeToDBBinary(struct SAMPLE *points, char *fileName, int fileType)；

函数功能：将 points 星表链表写入到数据库

输入：星表链表 points，星表链表所对应的文件名 fileName，文件类型 fileType，代表该文件是 reference 还是 sample

输出：将数据写入到数据库

3. 主程序输入参数

crossmatch v1.2 (2012 Nov 15)
usage: crossmatch [-method <plane>] [-errorRadius <20>] [-width <3096>] [...]
usage: crossmatch [-method <sphere>] [-errorRadius <0.00556>] [...]
-method <sphere|plane>: cross match method, using sphere coordinate or plane corrdinate
-width <number>: on plane corrdinate partition, the max value of X axis
-height <number>: on plane corrdinate partition, the max value of Y axis
-errorRadius <number>: the max error between two point, unit is degree, defalut 0.005556
-searchRadius <number>: the search area's radius, unit is degree, defalut equals errorRadius
-minZoneLength <number>: the min length of the zone's side, unit is degree, defalut equals errorRadius
-fitsHDU <number>: read fits file from the fitsHDU-th data area
-ref <path>: reference table path
-sample <path>: sample table path
-output <path>: output table path
-dbConfigFile <fileName>: file contain database config information.
Notice: -dbInfo is prior to -dbConfigFile
-dbInfo "<name1=value1,name2=value2...>": this option include the database configure information
host=localhost, IP Address or host name
port=5432, the port of PostgreSQL use
dbname=svomdb, database name
user=wanmeng, database user name
password= ,database user password
catfile_table=catfile_id
match_talbe=crossmatch_id
ot_table=ot_id
ot_flux_table=ot_flux_id
-show <all|matched|unmatched>:
all:show all stars in sample table including matched and unmatched
matched:show matched stars in sample table
unmatched:show unmatched stars in sample table

-terminal: print result to terminal
-cross: compare zone method with cross method, find the zone method omitted stars, and output to file
-processInfo: print process information
-fluxSDTimes <number>: the times of flux SD, use to filter matched star with mag
-h or -help: show help
-v or -version: show version number

example:

1: crossmatch -method sphere -errorRadius 0.006(20 arcsec) -searchRadius 0.018 -fitsHDU 2 -ref reference.cat -sample sample.cat -output output.cat -processInfo

2: crossmatch -method plane -errorRadius 10 -searchRadius 30 -width 3096 -height 3096 -fitsHDU 2 -ref reference.cat -sample sample.cat -output output.cat -processInfo

4. 附加功能——自动建表程序 CreateTableForCrossmatch

4.1. 程序功能

为方便交叉证认的数据库建表过程，我们对建表及删除表的命令进行封装，大大简化建表的过程。数据库表结构存储在文件中，每次建表的步骤如下：

1. 从配置文件中读取表结构；
2. 对表结构中的关键字符串（如表名等）进行替换；
3. 创建表。

4.2. 程序输入参数

程序默认输入文件是 create_table_database_config.txt，这个文件的名称可以通过-f 来指定。

通过配置文件中的信息，程序会找到 4 个表结构文件，默认是 table_struct 文件夹下的 catfile.txt, crossmatch.txt, ot.txt, ot_flux.txt。

默认程序的输出文件是 database_config.txt，该文件名可以通过命令-o 来指定，该文件作为 crossmatch 的输入文件。

注意：

1. 由于预先的表结构不可能预知将来建表时所使用的表名，索引表结构文件中有一些关键字是用来代指通用表名的，比如 catfile_table 是用来代表 catfile 相关的表，crossmatch_table 是用来代表 crossmatch 相关的表，ot_table 是用来代表 ot 相关的表，ot_flux_table 是用来代表 ot_flux 相关的表，这些关键字在实际建表之前，需要替换成对应的表名。
2. 目前的 CreateTableForCrossmatch 程序和 crossmatch 程序的耦合性比较强，当修改添加或减少表结构文件中的字段之后，需要修改 crossmatch 程序的代码，以使程序能够

正常运行，将来可以考虑把 crossmatch 程序的入库表结构也变成配置文件的形式，并与 CreateTableForCrossmatch 的保持一致。

程序的具体输入参数列表：

createtableforcrossmatch (2012 Nov 15)

usage: createtableforcrossmatch (defalut create today's tables use default config files)

or: createtableforcrossmatch [-c YYYYMMDD(today) | -d YYYYMMDD(today)] [-f configfile] [-o outfile]

--help or -h:	print help information
--fNumber or -fn <number>	table inherit, the number of file each table contained, must combine with --tNumber
--tNumber or -tn <number>	table inherit, the total number of table to be created, must combine with --fNumber
--create or -c <date>	<date> can be "YYYYMMDD" or "today", default "today" ， create four tables with name "prefix<date>" ， "prefix" read from configfile: "catfile_", "crossmatch_", "ot_", "ot_flux"
--delete or -d <date>	<date> can be "YYYYMMDD" or "today", default "today",delete four tables with name "prefix<date>" "prefix" read from configfile: "catfile_", "crossmatch_", "ot_", "ot_flux"
--configfile or -f <file name>	database config file name, default "create_table_database_config.txt"
--outfile or -o <file name>	config file for crossmatch program, default "database_config.txt"

example:

"createtableforcrossmatch -c today -f create_table_database_config.txt -o database_config.txt"

相关引用

[1] Cfitsio: http://heasarc.gsfc.nasa.gov/fitsio/

[2] Wcstools: http://tdc-www.harvard.edu/wcstools/

[3] 切平面：http://en.wikipedia.org/wiki/Gnomonic_projection

[4] 大圆距离：http://en.wikipedia.org/wiki/Great-circle_distance

[5] Postgresql 文档: http://man.ddvip.com/database/pgsqldoc-8.1c/libpq.html

[6] Jim Gray. The Zone Algorithm for finding Points-Near-a-Point or Cross-Matching Spatial Datasets.

[7] Alexander S. Szalay. Indexing the Sphere with the Hierarchical Triangular Mesh.

[8] MYSQL src download: http://download.softagency.net/MySQL/Downloads/

[9] MCS: http://ross.iasfbo.inaf.it/mcs

[10] DIF: http://ross.iasfbo.inaf.it/dif

附录一

1. 表结构及对应 SQL 语句

主表

```
create table catfile_id(
catid                bigserial,
catfile              varchar(25),
airmass             double precision,
magdiff             double precision,
jd                   double precision,
IS_REF               bool,
PRIMARY KEY(catid)
);
```

从表: 因为一个 catfile 对应多个 starid

```
create table crossmatch_id(
cid                  bigserial PRIMARY KEY,
starid               bigint,
crossid              bigint,
ra                   double precision,
dec                  double precision,
catid                bigint    REFERENCES catfile_table(catid) ON UPDATE CASCADE,
background          double precision,
classstar            double precision,
ellipticity          double precision,
flags                double precision,
mag                  double precision,
mage                 double precision,
magnorm              double precision,
fwhm                 double precision,
magcalib             double precision,
magcalibe           double precision,
pixx                 double precision,
pixy                 double precision,
thetaimage         double precision,
vignet              double precision,
pixx1                  double precision,
```

```
pixy1                  double precision,
fluxratio  double precision
);
CREATE INDEX crossmatch_table_crossid_index on crossmatch_table(crossid);
ANALYZE crossmatch_table;
CREATE INDEX crossmatch_table_catid_index on crossmatch_table(catid);
ANALYZE crossmatch_table;
```

crossid 和 catid 常用来“=”查询。

```
create table OT_id(
cid                    bigserial PRIMARY KEY,
starid                 bigint,
otid                bigint,
ra                     double precision,
dec                    double precision,
catid                  bigint    REFERENCES catfile_table(catid) ON UPDATE CASCADE,
background          double precision,
classstar              double precision,
ellipticity            double precision,
flags                  double precision,
mag                    double precision,
mage                   double precision,
magnorm                double precision,
fwhm                   double precision,
magcalib               double precision,
magcalibe              double precision,
pixx                   double precision,
pixy                   double precision,
inarea                 int,
thetaimage          double precision,
vignet                 double precision,
pixx1                  double precision,
pixy1                  double precision
);
CREATE INDEX ot_table_otid_index on ot_table(otid);
ANALYZE ot_table;
CREATE INDEX ot_table_catid_index on ot_table(catid);
ANALYZE ot_table;

Ot_flux_id(
cid                    bigserial PRIMARY KEY,
starid                 bigint,
crossid                bigint,
```

```
ra                  double precision,
dec                 double precision,
catid               bigint    REFERENCES catfile_table(catid) ON UPDATE CASCADE,
background          double precision,
classstar           double precision,
ellipticity         double precision,
flags               double precision,
mag                 double precision,
mage                double precision,
magnorm             double precision,
fwhm                double precision,
magcalib            double precision,
magcalibe           double precision,
pixx                double precision,
pixy                double precision,
thetaimage          double precision,
vignet              double precision,
pixx1               double precision,
pixy1               double precision,
fluxratio  double precision
);
CREATE INDEX ot_flux_table_otid_index on ot_flux_table(crossid);
ANALYZE ot_flux_table;
CREATE INDEX ot_flux_table_catid_index on ot_flux_table(catid);
ANALYZE ot_flux_table;
```

2. PostgreeSQL 数据插入方案

将数据写入到数据库主要有四种方式：

1. 直接每颗星信息使用一条 insert 语句，操作最简单，但是速度最慢；
2. 使用预编译函数 PQprepare 函数，也是每次插入一条语句，但是省去每次 SQL 语句的解析时间，速度慢；
3. 使用 COPY 指令文本方式，PostGreSQL 数据库的 COPY 指令适合大批量插入数据时使用，通过名字 COPY 可以看出，其操作就像直接将数据拷贝到数据库。使用该方式时需要将各种基本数据转换为文本后，然后将文本串传给 COPY 指令进行插入操作，速度不够快，原因是将基本数据转换为文本需要消耗大量的时间，而且将文本数据传入到数据库中之后，又需要将文本数据转换为基本数据。
4. 使用 COPY 指令二进制方式，直接将基本数据以二进制形式插入到数据库，速度非常快。下面将简要介绍该方式。

3. PostGreSQL 数据库 COPY 命令 C 语言环境下使用注意事项

1. COPY 命令最常用的是输入输出到文件，同时 COPY 的输入输出也可以是标准输入输出或内存空间，不过无论是那种方式，使用二进制的 COPY 命令时，数据组成格式都是一样的。若刚开始对数据的组成格式比较困惑，可以先用 COPY 命令从数据库导出几行数据到文件，然后对文件中的二进制数据加以分析，之后或许会豁然开朗。为了方便说明，将以文件的形式进行举例。
2. 数据头，二进制文件需要有个文件头，长度 19 个字节，其中前 11 个字节内容为“PGCOPY\n\377\r\n\0”的二进制形式，后 8 个字节都是 0。
3. 数据部分，数据部分是由一行一行记录组成的，数据部分的每行之间没有间隙，每行挨着一行。每行的的前两个字节为该行的数据列个数，后面依次跟着这么多的数据列，每列的格式是：4 字节的该列数据的长度，加上该列数据的二进制形式。列数据的长度，也就是该列的数据类型的长度，例如 double 类型的数据长度为 8。每列之间没有空隙。
4. 列数据的二进制形式，小于 32 位的数据类型可以直接通过内存拷贝来获取二进制形式，大于等于 32 位的数据类型需要使用 htonl 函数进行将数据转换为网络字节编码形式，htonl 函数的输入为长度 32 位的 unsigned int 类型，所有大于 32 位的数据需要拆分为多个 32 位的数据。举例来说，double 类型长度为 8 字节 64 位，需要拆分为两个 unsigned int 的数据分别使用 htonl 函数进行转换。
5. 转换大于 32 位的数据，不同操作系统的内存数据存储结构有可能不同，在对数据进行转换时要清楚操作系统是按高字节存储还是按低字节存储。
6. PostGreSQL 数据库中有 BigInt 数据类型，其长度为 8 字节 64 位，使用时应该与 32 位的 Int 类型区分开。
7. 结尾，文件结尾需附加两字节的 0，据经验也可以不加。

4. PostgreSQL 二进制 COPY 程序示例

```
void writeToDBBinary(struct SAMPLE *sample) {

    if (sample == NULL) return;

    char *host = "localhost";
    char *port = "5432";
    char *options = NULL;
    char *tty = NULL;
    char *dbname = "svomdb";
    char *user = "wanmeng";
    char *pwd = NULL;
    PGconn *conn;
    PGresult *pgrst;
```

```
conn = PQsetdbLogin(host, port, options, tty, dbname, user, pwd);
if (PQstatus(conn) == CONNECTION_BAD) {
        fprintf(stderr, "connect db failed! %s\n", PQerrorMessage(conn));
        PQfinish(conn);
        return;
}

char *sql = "COPY test_xy(id_s,ra_s,dec_s,id_r,ra_r,dec_r) FROM STDIN WITH BINARY"; //
char *buf = (char*) malloc(MAX_BUFFER * sizeof (char));
int i = 0;
char *sendHeader = "PGCOPY\n\377\r\n\0";
pgrst = PQexec(conn, sql);
if (PQresultStatus(pgrst) == PGRES_COPY_IN) {
        struct SAMPLE *tSample = sample->next;
        unsigned short fieldNum = 6;
        struct strBuffer strBuf;
        strBuf.data = buf;
        strBuf.len = MAX_BUFFER;
        strBuf.cursor = 0;
        memcpy(strBuf.data, sendHeader, 11);
        strBuf.cursor += 11;
        unsigned int zero = '\0';
        memcpy(strBuf.data + strBuf.cursor, (char*) &zero, 4);
        strBuf.cursor += 4;
        memcpy(strBuf.data + strBuf.cursor, (char*) &zero, 4);
        strBuf.cursor += 4;

        while (tSample) {
                addInt16(&strBuf, fieldNum);
                addInt32(&strBuf, tSample->id);
                addFloat8(&strBuf, tSample->alpha);
                addFloat8(&strBuf, tSample->delta);
                addInt32(&strBuf, tSample->reference->id);
                addFloat8(&strBuf, tSample->reference->alpha);
                addFloat8(&strBuf, tSample->reference->delta);

                int copydatares = PQputCopyData(conn, strBuf.data, strBuf.cursor);

                tSample = tSample->next;
                strBuf.cursor = 0;
        }
        PQputCopyEnd(conn, NULL);
}
```

```
    PQclear(pgrst);
    PQfinish(conn);
    free(buf);
}
```

附录二——软件待完善的功能

1. 直接经纬度分区的问题：

直接经纬度分区算法中的分区面积差异太大，最小面积（极点附件）与最大面积（赤道附件）相差近百倍，而且从赤道到极点按纬度方向，分区的面积是逐渐减小的。由于极点附件（80~90 度）每个分区的面积过小，所以在进行交叉匹配操作时点的待匹配区域可能并不像图 3 所示的四种情况。

解决方案：对分区算法进行改进，不一定要等面积分区，但是要保证极点附件（80~90 度）的分区面积要足够大，比如可以考虑按纬度递增适当增加分区间隔，或者直接按纬度分段分区等等。

2. HTM 分区的问题：

采用 HTM 方式比直接经纬度分区方式的错误率更低，但是速度更慢。比如在 reference 文件为 IPsao5946-540.Fcat 有 8191 颗星，sample 文件为 IPsao5946-115.Fcat 有 9843 颗星，HTM 方式的匹配时间为 0.27，直接经纬度分区方式的匹配时间为 0.01。HTM 分区算法直接使用 HTM 库进行分区索引的计算，使用 DIF 库来查找点的邻近分区，估计是计算 HTM 索引和点的邻近分区比较复杂，带来计算量的增加，从而影响程序的性能。

解决方案：对 HTM 索引算法和点的邻近区域查找算法进行调优。

3. 建立分区索引时的问题：

在向分区中添加星的时候，程序中对每颗星安装 ra 从小到大排序，在交叉匹配时结合这个排序信息，会大大减少匹配的个数。具体是当 sample 星与 reference 分区链表中的星满足条件：(reference->ra + SUBAREA > sample->ra) && (reference->ra - SUBAREA < sample->ra)时才进行比较。但是这个过滤条件会导致最终的匹配结果不足。

解决方案：适当选取 SUBAREA 的值，SUBAREA 稍微偏大比较好，但是 SUBAREA 过大就失去过滤的作用。

4. GPU 实现时的分区问题：

GPU 实现交叉匹配主要面向超大星表匹配，这时在建立分区索引时不用排序。

5. 数据库入库问题

单个表的记录条数在逐渐增加的过程中，数据的入库时间是逐渐增加的，当超过几百万

到几千万时，插入一组数据（2W 到 3W 条记录）会达到好几秒。

对数据库进行不同的设置会导致不同的性能差异。具体表现：

1. 在建表时，a，有主外键有索引，b，无主外键有索引，c，无主外键无索引，这三种情况的性能有 a 和 b，c 有明显的差别。
2. 对数据库属性的设置，对应文件/var/lib/pgsql/9.1/data/postgresql.conf：fsync = off(默认 on), full_page_writes = off(默认 on), checkpoint_segments = 50(默认 3)，对性能也有明显的影响。
3. 对数据库随着数据条数的增加，插入时间线性增加的现象，我们可以采用 PostgreSQL 的多级继承分表机制，控制表的大小，使每个表的记录数不超过几百万。

对下面的表和图中的标识符进行说明：

noIndex：建表无主外键无索引

haveIndex：建表无主外键有索引

haveForeign：建表有主外键有索引

on：数据库参数优化（fsync = off, full_page_writes = off, checkpoint_segments = 50）

off2：数据库参数不优化（fsync = on, full_page_writes = on, checkpoint_segments = 3）

表 1 优化参数开启或关闭时的三种建表方式的 1001 个文件入库时间统计表

		Total(s)	avg(s)	max(s)	min(s)
off2	noIndex	867.239	0.866373	4.685	0.358
	haveIndex	27518.29	27.4908	82.646	0.711
	haveForeign	28433.82	28.40542	96.707	0.841
on	noIndex	702.044	**0.70134**	3.164	0.172
	haveIndex	**744.768**	**0.74402**	**7.806**	**0.245**
	haveForeign	1685.006	1.683323	7.902	0.537

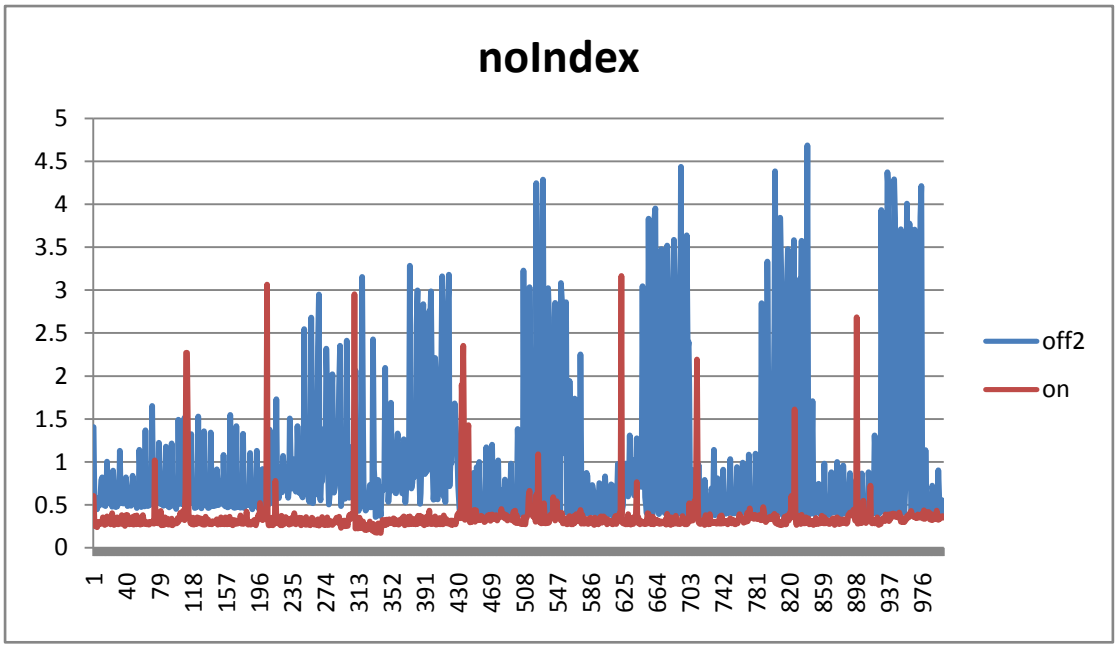

图 8 建表无主外键无索引时的数据库优化参数开启或关闭下的时间统计

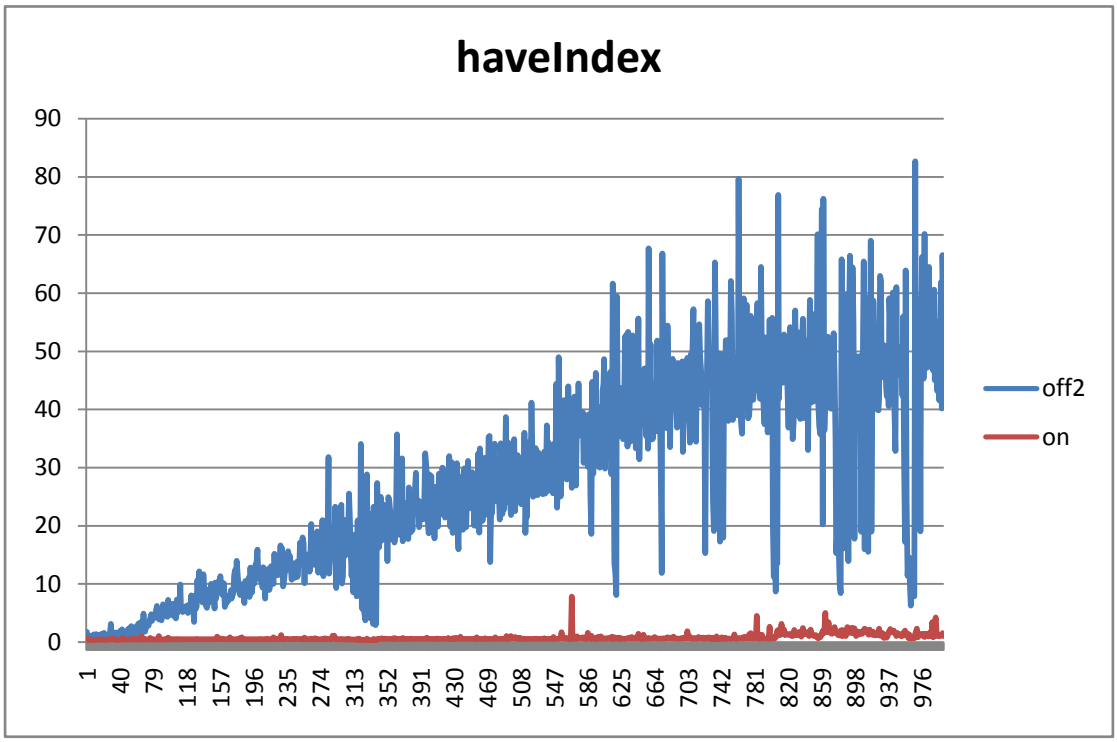

图 9 建表无主外键有索引时的数据库优化参数开启或关闭下的时间统计

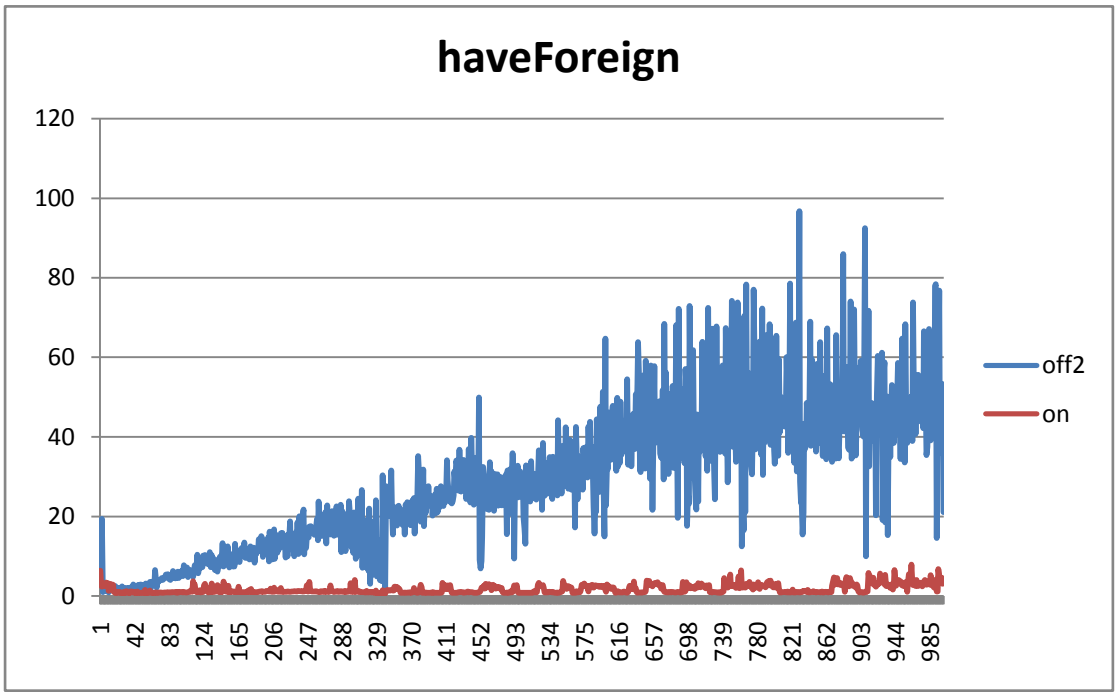

图 10 建表有主外键有索引时的数据库优化参数开启或关闭下的时间统计

6. 程序的整体调度问题

整的上面章节提到的第三条（多级继承分表机制），现在的问题是，分表机制需要制定详细的分表规则，即具体指定每个表存放第多少条到多少条记录：

1. 不能通过自增主键 cid 来出发规则，对 crossmatch_id 表中的每条记录只能通过 catid 来进行分区，即指定 catid 文件中的所有记录存放到指定的表中；

2. COPY 命令不能触发 postgreSQL 的规则，官网说可以触发触发器，即对每条插入记录都通过气 catid 判断属于哪个分区，这个肯定会影响性能；
3. 通过在程序当中指定一个计算器，来有程序判断应该往哪个分区插入数据，以及是否创建分区，这样就会增加程序与数据库的耦合度。

如果采用第三种方式，则应该从整体的角度设计整个程序，比如程序长期驻守内存，使用多线程异步插入（一个线程交叉证认，一个线程入库）等等。