# Congestion拥塞控制源码解读

## 文档编写者：曹泽敏

## 一、配置文件参数解释

## congestion.config

配置格式

primary\_destination=value secondary\_specifier=value tag=value

<1> primary\_destination 针对匹配上的请求生效此行配置，是必选项，且每行配置此配置项仅允许配置一次，支持选项如下

dest\_domain: A requested domain name. 域名

dest\_host: A requested hostname. 主机名

dest\_ip: A requested IP address. 主机IP

url\_regex: A regular expression (regex) to be found in a URL. 通过URL的正则表达式

<2> secondary\_specifier 是可选项，支持多选项配置，但同一个配置项不能重复配置

port: A requested URL port or range of ports. 请求的端口

prefix: A prefix in the path part of a URL. 请求URL路径的前缀

<3>tag标签，未配置的则选择默认值

max\_connection\_failures

Default: 5 The maximum number of connection failures allowed within the fail window described below before Traffic Server marks the origin server as congested.

在某个周期内，连接源站的失败次数达到最大次数则标记为拥塞

fail\_window

Default: 120 seconds. The time period during which the maximum number of connection failures can occur before Traffic Server marks the origin server as congested.

指 max\_connection\_failures 的统计窗口周期值

proxy\_retry\_interval

Default: 10 seconds. The number of seconds that Traffic Server waits before contacting a congested origin server again.

指源站拥塞时的重试连接间隔时间

client\_wait\_interval

Default: 300 seconds. The number of seconds that the client is advised to wait before retrying the congested origin server.

源站拥塞，告诉客户端多久后进行重试

wait\_interval\_alpha

Default: 30 seconds. The upper limit for a random number that is added to the wait interval.

此选项与 client\_wait\_interval 一起合用，来计算告诉客户端进行重试的时间，主要是错开客户端之间的访问，尽量避免多个客户请求同时来请求

客户端重试时间的计算方法为： client\_wait\_interval + 随机数% wait\_interval\_alpha + 源站拥塞时剩余的重试连接源站的时间

live\_os\_conn\_timeout

Default: 60 seconds. The connection timeout to the live (uncongested) origin server.

If a client stops a request before the timeout occurs, then Traffic Server does not record a connection failure.

源站未拥塞时，请求的超时时间

live\_os\_conn\_retries

Default: 2 The maximum number of retries allowed to the live (uncongested) origin server.

源站未拥塞时，请求允许的重试连接次数

dead\_os\_conn\_timeout

Default: 15 seconds. The connection timeout to the congested origin server.

源站拥塞时，请求的超时时间

dead\_os\_conn\_retries

Default: 1 The maximum number of retries allowed to the congested origin server.

源站拥塞时，请求允许的重试次数

关于超时和重试： 当源站不管是否拥塞，只是限制请求数，此时还是可以回源的。

max\_connection

Default: -1 The maximum number of connections allowed from Traffic Server to the origin server.

与源站的最大连接数，超过则表示拥塞

error\_page

Default: "congestion#retryAfter" The error page sent to the client when a server is congested. You must enclose the value in quotes;

返回给客户端的错误页面

congestion\_scheme

Default: "per\_ip" Specifies if Traffic Server applies the rule on a per-host ("per\_host") or per-IP basis ("per\_ip"). You must enclose the value in quotes.

For example:

if the server www.host1.com has two IP addresses and you use the tag value "per\_ip",

then each IP address has its own number of connection failures and is marked as congested independently.

If you use the tag value "per\_host" and the server www.host1.com is marked as congested, then both IP addresses are marked as congested.

只允许 per\_host 和 per\_ip 两种值， per\_host 表示源站拥塞统计通过域名， per\_ip 表示源站拥塞统计通过IP。当源站有多个服务器时，建议用 per\_ip

Notes：

拥塞控制主要分两类：

<1> 一种是针对当前最大连接数的

<2> 二种是针对连接重试失败超时的

## records.config

拥塞功能生效，必须打开此开关

CONFIG proxy.config.http.congestion\_control.enabled INT 1

congestion.config中的TAG 可以通过下面配置修改默认值

CONFIG proxy.config.http.congestion\_control.default.tag INT|STRING value

## 拥塞时拒绝的请求数统计查看

查看源站响应失败超时拥塞时被拒绝的请求数

./traffic\_line -r proxy.process.congestion.congested\_on\_conn\_failures

查看连接数超过最大时被拒绝的请求数

./traffic\_line -r proxy.process.congestion.congested\_on\_max\_connection

Notes： 此统计是针对所有域名

## 二、数据结构：

### 配置模块

此模块与 cache.config 配置模块一样，所有请求都必须匹配 congestion.config 中配置的所有配置项

具体代码实现若有兴趣，可以另外探讨

### 域名数据存储模块

类功能简介：

CongestionControlRecord: 保存congestion.config 的配置，一个对象对应一条规则

CongestionMatcherTable: 所有的配置信息 CongestionControlRecord 都保存在此类中

FailHistory: 保存与源站连接失败的相关信息

CongestionEntry: 保存着拥塞状态、当前连接数和失败重试连接等信息，CongestionEntry保存一个 CongestionControlRecord 指针，多对一关系，

一个 CongestionEntry 对应一个 FailHistory

CongestionDB: 用二级HASHTABLE保存着所有的 CongestionEntry 信息

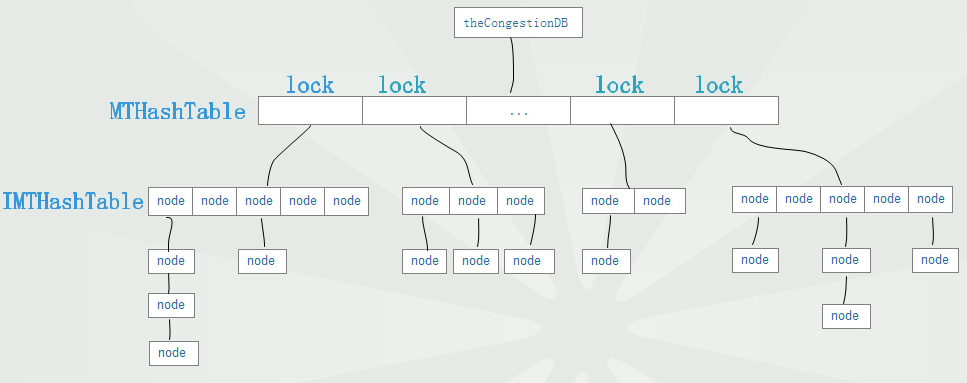
每类请求都有一个 CongestionEntry 对象来进行存储

1、每个 CongestionEntry 对象都对应一个key，key的计算参数如下：

源站域名或IP、端口port、前缀prefix（URL的前缀）

请求的这三者相同则作为一类请求，通过计数器共用一个 CongestionEntry 对象

2、CongestionDB 的HASHTABLE的数据结构如下：



I、 CongestionEntry 对象保存在 node 结点中

II、所有的 CongestionEntry 通过上述HASH结构保存到 theCongestionDB 指针指向的 CongestionDB 对象中

III、MTHashTable 的table长度是固定的，每个元素对应一把lock锁，其数目为 MT\_HASHTABLE\_PARTITIONS

IV、IMTHashTable HASH是通过拉链表方式解决冲突的。可以动态扩容，扩容的标准是每个 IMTHashTable 表的平均链表长度超过 MT\_HASHTABLE\_MAX\_CHAIN\_AVG\_LEN 时，对该HASH表进行一次扩容调整，扩容前会删除过期失效的 CongestionEntry 结点

V、每次查询和插入 CongestionEntry 时，必须先获取 MTHashTable 中对应的互斥锁lock

## 三、函数处理流程

### （1）初始化

Congestion.config在启动时通过 CongestionMatcherTable::reconfigure 将所有配置解析到 CongestionMatcher 指针指向的 CongestionMatcherTable 对象中

并通过 revalidateCongestionDB 实例化 CongestionDB 对象，用 theCongestionDB 指针将其保存

### （2）拥塞访问流程

1、 每个回源请求都会调用 HttpSM::do\_congestion\_control\_lookup 通过 get\_congest\_entry 函数获取该请求的 t\_state.pCongestionEntry

2、 HTTP请求先通过 CongestionControlled 查找对应的 congestion.config 配置项规则 CongestionControlRecord

3、 再利用 源站域名或IP、端口port、匹配规则CongestionControlRecord 通过 make\_key 计算对应的key值

4、 通过key值先获取 MTHashTable 对应的lock，查找或插入 CongestionEntry ，并返回对应的 CongestionEntry 对象来进行拥塞判断

### （3）相关变量和函数解释

t\_state 中与拥塞控制相关的参数

// congestion control

CongestionEntry \*pCongestionEntry;

StateMachineAction\_t congest\_saved\_next\_action; //下一状态保存，有 HttpTransact::STATE\_MACHINE\_ACTION\_UNDEFINED, HttpTransact::ORIGIN\_SERVER\_OPEN 和

// HttpTransact::ORIGIN\_SERVER\_RAW\_OPEN 这三个值，主要用于在拥塞状态下的状态跳转

int congestion\_control\_crat; // 'client retry after',这个主要是影响访问日志的记录

int congestion\_congested\_or\_failed; //表示此请求是否拥塞或失败，主要用于对 t\_state.pCongestionEntry->go\_alive(); 即 m\_congested 状态更新为 0

int congestion\_connection\_opened; //表示此请求的拥塞控制功能是否打开，主要用于 pCongestionEntry->connection\_closed(); 即 m\_num\_connections 值的更新

FailHistory 类

成员变量

long last\_event; //上次失败事件注册时间

int events; //当前失败的事件总个数

成员函数被调顺序

HttpTransact::handle\_parent\_died --> CongestionEntry::failed\_at --> FailHistory::regist\_event --> FailHistory::init\_event

CongestionEntry::CongestionEntry --> CongestionEntry::clearFailHistory --> FailHistory::init

CongestionControlEnabledChanged --> revalidateCongestionDB --> CongestionDB::revalidateBucket --> CongestionEntry::validate --> CongestionEntry::applyNewRule --> CongestionEntry::init --> CongestionEntry::clearFailHistory --> FailHistory::init

此类通过 max\_connection\_failures 和 fail\_window 参数来控制拥塞

CongestionEntry 类

成员变量

// State -- connection failures

FailHistory m\_history; //保存 CongestionEntry 一定时间内访问源站失败的记录

ink\_hrtime m\_last\_congested; //上次失败事件的访问时间，由 m\_last\_congested = m\_history.last\_event; 赋值，用于日志打印

volatile int m\_congested; //0 | 1 当前拥塞判断值，主要是与 m\_history 相关的拥塞开关

int m\_stat\_congested\_conn\_failures; //通过 m\_congested 开关拥塞失败连接的总次数

volatile int m\_M\_congested; //连接数超过限制时的拥塞开关

ink\_hrtime m\_last\_M\_congested; //上次连接数超过限制时被拥塞的时间

// State -- concorrent connections

int m\_num\_connections; //当前连接源站的总连接数

int m\_stat\_congested\_max\_conn; //与 m\_M\_congested 开关相关，连接数超过限制时被限制的总连接数

注: m\_congested 的放开是受 m\_congested, m\_M\_congested 和源站是否挂掉了一起控制的

成员函数

virtual RD\_Type data\_type(void); //主要用于配置项查找 CongestionControlRecord::UpdateMatch

inline bool proxy\_retry(ink\_hrtime t); //通过当前时间、 m\_history.last\_event 、 proxy\_retry\_interval 来判断是否可以重试

inline int client\_retry\_after(); //返给客户端时让其重试的时间，其值是 proxy\_retry\_interval, m\_history.last\_event, 当前时间, client\_wait\_interval, wait\_interval\_alpha 一起计算出

inline int connect\_timeout(); //通过判断 m\_congested 返回 dead\_os\_conn\_timeout 或 live\_os\_conn\_timeout

inline int connect\_retries(); //通过判断 m\_congested 返回 dead\_os\_conn\_retries 或 live\_os\_conn\_retries

## 四、应用场景

此拥塞模块主要是针对 回源连接数 和 源站响应失败超时 这两种情况进行拥塞控制

缺点：

1、请求匹配配置文件时，需要将所有配置项都要匹配一遍（不管是host,ip,domain还是regex），与cache.config的配置差不多。若配置多则很影响性能。