Introdução a NoSQL e escalabilidade de dados

Por Eugene Ciurana

Conteúdo:

Introdução

O cartão de referência nº 43 da DZone é uma introdução a terminologias e técnicas de sistemas de alta disponibilidade e escalabilidade (https://dzone.com/refcardz/scalability). O próximo passo lógico é o manuseio escalável de massivos volumes de dados resultantes destas capacidades de processamento.
Este cartão de referência, desmistifica NoSQL e as técnicas de escalabilidade de dados com a introdução de alguns conceitos fundamentais. Oferece também uma visão das tecnologias atuais disponíveis nesta área e de como utiliza-las.

O que é escalabilidade de dados?

Escalabilidade de dados é a capacidade de um sistema armazenar, manipular e analisar quantidades cada vez maiores de dados sem a redução de sua disponibilidade, desempenho ou a taxa de transferência.
A Escalabilidade de dados é conseguida pela combinação de processamentos mais poderosos e mecanismos de armazenamento maiores e mais eficientes.
Bancos de dados relacionais e hierárquicos escalam adicionando-se mais processadores, capacidade de armazenamento, sistemas de cache e outros. Logo eles atingem um custo ou um limite de escalabilidade que tornam difícil ou impossível prosseguir.
Estes sistemas de gerenciamento de banco de dados foram projetados como unidades únicas e para manter a integridade dos dados impondo um esquema de regras para garanti-las. Isto garante a escalabilidade vertical, mas não horizontal.

Dica Quente

RAC Oracle é um cluster de múltiplos computadores que acessam uma base de dados comum. Isto é escalabilidade vertical porque o processamento (comumente na forma de store procedures) pode ser escalável, mas as facilidades de armazenamento compartilhado não escalam com o cluster.

Integridade de dados e esquemas são adequados para dados normalizados, uniformes e transacionais. Para dados não estruturados ou de rápido crescimento isto pode ser difícil ou gerar altos custos.

Dica Quente

Replicação de dados não é o mesmo que escalabildade de dados

Arquitetura de dados escalável

Existem dois tipos gerais de arquitetura que são utilizadas para a construção de sistemas de dados escaláveis: Data Grids ou NoSQL. A implementação de ambos frequentemente compartilham as mesmas características.

Data Grids

Data Grids processam as cargas de trabalho como serviços independentes que não necessitam compartilhar os dados entre os processos. O sistema da rede ou de armazenamento pode ser compartilhado por todos os nós d rede, mas os resultados intermediários não interferem nos demais processamentos ou demais nós da rede, como ocorre em cluster que utiliza MapReduce.

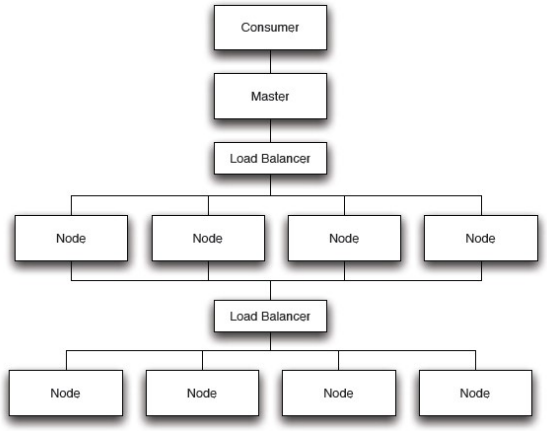

Figura 1 – Data Grid

Os Data Grids utilizam uma única API (como um Web Service ou nativo da linguagem de programação) que abstrai a topologia e implementação do consumidor do processamento dos dados.

Áreas de aplicação

- Modelagem financeira
- Mineração de dados
- Análise de Click Stream
- Clusterização de documentos
- Classificação ou Grepping distribuídos
- Simulações
- Construção de índex invertidos
- Desdobramento de proteínas

NoSQL

Pode ser descrito como um banco de dados não relacional, que permite escalabilidade horizontal com suporte a replicação. Os aplicativos interagem por meio de uma API e os dados são armazenados "livres de esquemas" em simples repositórios, geralmente utilizando grandes arquivos ou blocos de dados. O repositório geralmente é um sistema de arquivos concebido para executar operações NoSQL.

Dica Quente

NoSQL é uma abreviação de "não apenas SQL". Arquiteturas completas quase sempre utilizam banco de dados tradicionais e NoSQL

A configuração NoSQL são melhores para aplicações "não-OLTP" e processam grandes volumes de dados estruturados e não estruturados a um menor custo e com maior eficiência do que os RDBMs e stored procedures.

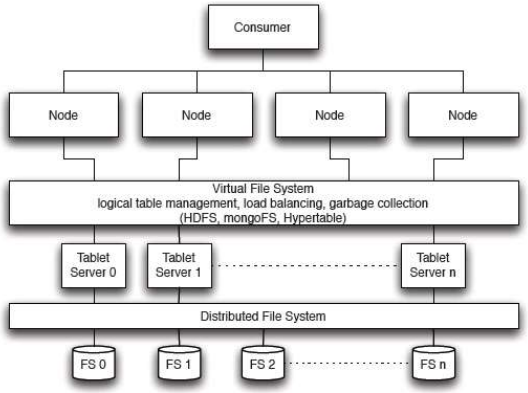

Figura 2 – Topologia NoSQL

Sistemas NoSQL possuem alta replicação (o commit não ocorre até o dado ser gravado com sucesso em no mínimo em 2 locais distintos) e os sistemas de arquivos são otimizados para commits de "write-only". Os dispositivos de armazenamento são formatados para utilizar grandes blocos de dados (32 MB ou mais). Cache e buferização são utilizados para um alto volume de I/O. O banco de dados NoSQL é implementado como uma rede para o processamento de MapReduce, queries, CRUD, etc ...

Áreas de aplicação

- Armazenamento de documentos
- DB de objetos
- DB de Grafos
- Armazenamento Chave/Valor
- Eventualmente consistem Chave/Valor armazenado
- Simulações
- Construção de índex invertidos
- Desdobramento de proteínas

Esta lista é uma contrapartida para as áreas de aplicação dos Data Grids: os dados "alimentam" a rede e juntos formam um banco de dados NoSQL.

Analogia entre NoSQL x RDBMS x OO

NoSQL	RDBMS	OO
Coleção	Tabela	Classe
Documento	Registro	Objeto
Atributo	Coluna	Propriedade

NoSQL é para você ?

Preparação:

- Não seja vítima da moda NoSQL
- Não seja teimoso: nem NoSQL, nem RDBMs podem ser utilizados em todos os casos
- Utilize o teorema CAP em seus casos de uso para determinar a viabiliade

Teorema Brewer's (CAP)

É impossível para um sistema de computação distribuída implementar todas as três garantias:

- Consistência(**C**onsistency): todos os nós enxergam o mesmo dado ao mesmo tempo

- Disponibilidade(**A**vailability): falha em um nó não impedem que os demais continuem operando
- Tolerância a falhas (**P**artition tolerance): a rede continua a funcionar mesmo após a falha em um nó da rede

Uma vez que apenas duas destas características são garantidas para qualquer sistema escalável, utilize suas especificações funcionais e o SLA de negócios (acordo do nível de serviço) para determinar quais são suas metas mínimas e metas CAP, escolhendo duas delas para atendimento de suas necessidades e então prossiga com a tecnologia escolhida.

Regra geral

O principal objetivo do NoSQL é escalabilidade horizontal. Isto é possível pela redução da semântica transacional e da integridade referencial

A figura 3 mostra a melhor correspondência entre os requisitos CAP de um aplicativo e os sistemas SQL e NoSQL sugeridos.

Figura 3 – Gráfico de seleção CAP
Fonte (*Nathan Hurst's Blog*)

Mongo DB

Mongo DB é um banco de dados baseado em documentos situado entre banco de dados que utilizam tecnologia chave/valor como Datastore e Mencache DB, e as facilidades de consulta e robustes dos RDBM's. Algumas de suas principais características incluem:

- Armazenamento orientado a documentos. Os dados são manipulados como documentos JSON
- Consultas. Utiliza JavaScript e possui APIs para execução das consultas nas principais linguagens de programção.
- Atualização no local. Atomicidade
- Indexação. Qualquer atributo de um documento pode ser usado para indexação e otimização nas consultas.
- Auto fragmentação. Possibilita a escalabilidade horizontal.
- Map/Reduce. Um cluster MongoDB permite executar pequenos Jobs MapReduce que um cluster Hadoop com significantes menores custos e mais eficiência.

Mongo DB possui o seu próprio sistema de arquivos que otimiza operações de I/O dividindo grandes objetos em partes pequenas. Documentos são armazenados em duas coleções separadas: arquivos contendo os metadados e partes que formam um documento maior quando combinados com as informações de armazenamento do banco de dados. Sua API disponibiliza funções para manipulação dos arquivos, partes e índices. Possui ferramenta de administração que possibilita a manutenção de GridFS.

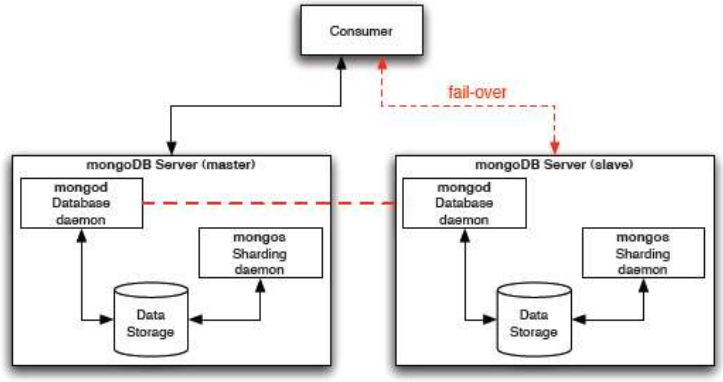

Figura 4 – Cluster MongoDB

Um cluster MongoDB consiste das partes mestre e escravo. Se necessário o escravo pode tornar-se mestre em um cenário livre de falhas. A configuração mestre/escravo (também conhecida como Ativo/Passivo cluster A/P) ajuda a garantir a integridade dos dados, uma vez que somente o mestre pode realizar as alterações em qualquer momento.

Um commit só é bem sucedido se os dados forem gravados na GridFS e replicados nos escravos.

> Dica quente
>
> Mongo DB possui um limite na configuração mestre/escravo. É útil apenas para inserções, consultas e exclusões de determinados objetos. Não deve ser utilizado em atualizações que um único objeto possa ocorrer simultaneamente.

Caching

Mongo DB possui sistema de cache interno e que roda diretamente no cluster sem a dependência de recursos externos. Qualquer consulta é submetida ao cache na RAM otimizando as taxas de transferência e reduzindo o I/O.

Formato Documento

Mongo DB mantem os documentos no formato BSON, que é uma codificação binária do JSON. Foi concebido para ser móvel, leve e eficiente. As aplicações podem mapear documentos BSON/JSON de modo nativo como dicionários, listas e arrays deixando sua interpretação para o drive do MongoDB utilizado pela linguagem.

Exemplo BSON

```
{
  'name' : 'Tom',
  'age' : 42
}
```

Language	Representation
Python	`{ 'name' : 'Tom', 'age' : 42 }`
Ruby	`{ "name" => "Tom", "age" => 42 }`
Java	`BasicDBObject d; d = new BasicObject(); d.put("name", "Tom"); d.put("age", 42);`
PHP	`array( "name" => "Tom", "age" => 42);`

As linguagens dinâmicas oferecem um mapeamento mais próximo a BSON/JSON que as linguagens compiladas. A especificação completa sobre BSON esta disponível em: http://bsonspec.org/

Programação MongoDB

A programação em MongoDB necessita de um servidor ativo rodando o MongoD, os demons dos databases (figura 4) e uma aplicação cliente em uma linguagem que utilize os drivers.

> **Dica quente**
>
> Todos os exemplos deste cartão de referência foram escritos em Phyton para facilitar.

Iniciando o servidor:

Log no servidor mestre e execute o seguinte comando:

[servername:user] ./mongod

O servidor irá mostrar mensagens de status na console de saída padrão.

Exemplo de programação

Este exemplo aloca um banco de dados, mesmo inexistente, instancia uma coleção no servidor e executa um conjunto de queries. A documentação para desenvolvedores esta disponível em: http://www.mongodb.org/display/DOCS/Manual

```
#!/usr/bin/env jython

import pymongo

from pymongo import Connection

connection = Connection('servername', 27017)

db = connection['people_database']

peopleList = db['people_list']

person = {
  'name' : 'Tom',
  'age'  : 42 }
```

```
peopleList.insert(person)

person = {
  'name' : 'Nancy',
  'age'  : 69 }

peopleList.insert(person)

# find first entry:
person = peopleList.find_one()

# find a specific person:

person = peopleList.find_one({ 'name' : 'Joe'})

if person is None:
  print "Joe isn't here!"
else:
  print person['age']

# bulk inserts
persons = [{ 'name' : 'Joe' }, {'name' : 'Sue'}]

peopleList.insert(persons)

# queries with multiple results

for person in peopleList.find():
  print person['name']

for person in peopleList.find({'age' : {'$ge' : 21}}).sort('name'):
  print person['name']

# count:

nDrinkingAge = peopleList.find({'age' : {'$ge' : 21}}).count()

# indexing

from pymongo import ASCENDING, DESCENDING

peopleList.create_index([('age', DESCENDING), ('name', ASCENDING)])
```

A documentação sobre PyMongo esta disponível em: http://api.mongodb.org/python - Guias aonde pode ser encontrado suporte para outras linguagens.

O programa exemplo mostrado acima, executa os seguintes passos:

- Conecta-se ao banco de dados iniciado no comando anterior.
- Executa um attach ao banco de dados. Observe que o mesmo é tratado como um array
- Obtem uma coleção (equivalente a uma tabela RDBM's)
- Insere uma ou mais entidades
- Consulta uma ou mais entidades

Embora o MongoDB trate internamente todos esses dados como BSON, a maioria das APIs permite o uso no estilo dicionário de forma a agilizar.

Objeto ID

Uma inserção bem sucedida no banco de dados resulta em um ID do objeto. Este identificador é exclusivo do objeto dentro do banco de dados. Ao executar uma consulta um valor é retornado para este atributo:

```
{
  "name" : "Tom",
  "age" : 42,
  "_id" : ObjectId('999999')
}
```

O Usuário pode substituir este atributo por outro, desde que seja único, ou permitir que o MongoDB faça isto automaticamente.

Casos de uso

- Cache – mais robusto alem da persistência, quando comparado com sistemas puros de cache.
- Alto volume de processamento – RDBM's são muito mais caros e menos ágeis se comparados.
- Armazenamento de dados e programas JSON – alguns Web Services RESTful utilizam dados JSON; eles podem ser armazenados no MongoDB sem a necessidade da transliteração (especialmente quando comparado com documentos XML)
- Sistema de gerenciamento de conteúdo – objetos JSON/BSON podem representar qualquer tipo de documento incluindo aqueles que binários.

Desvantagens

- Não permite JOINs – cada documento é único
- Queries complexas – algumas queries complexas e índices tem um melhor tratamento por RDBMs.
- Inexistência de lock a nível de registro – inadequado para transações que necessitam de garantia de transação.

Se alguma destas referências é melhor para sua aplicação, talvez um banco de dados SQL seja melhor para você.

GIGASPACES XAP

A plataforma GigaSpaces eXtreme Aplication é uma rede concebida para substituir os tradicionais servidores de aplicação. Ela opera baseada no modelo de processamento de eventos aonde a aplicação envia os objetos para os nós da rede associados a uma determinada partição de dados. O sistema pode ser configurado para que o estado do dado dispare uma

trigger ou uma aplicação execute uma série de commandos. GigaSpaces XAP tambem gerencia todos os aspectos de operação e o pool de threading e conexões. Ainda implementa transações Spring e auto-recovery. O sistema detecta qualquer falha de operação de um nó e realiza o roll back automaticamente e em seu lugar coloca o processamento em outro nó para completá-lo.

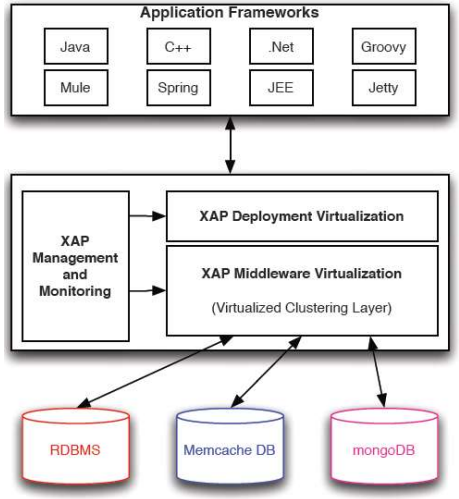

Figura 5 – Plataforma GigaSpaces XAP

GigaSpaces disponibiliza persistência de dados, processamento distribuído e sistema de cache junto a banco de dados SQL e NoSQL. A API GigaSpaces encapsula todas as operações de back-end (execução de jobs, persistência de dados e cache) tornando isto transparente para a aplicação. Implementa operações de computação distribuída como MapReduce e realiza a execução nos nós, ou dispara para subsistemas sujacentes disponíveis, se a funcionalidade estiver disponível (ex.: mongoDB, Hadoop). O aplicativo pode utilizar controle de transações usando a API Spring, o controle transactional GigaSpaces XPA, ou pela implmentação de fluxos de trabalhos aonde o armazenamento NoSQL utiliza um tipo ou um grupo de tipos. Isto pode ser realizado explicitamente em sistemas como mongoDB, mas tambem pode utilizar outros sistemas NoSQL como Google APP Engine.

Programação GigaSpaces XAP

A API GigaSpaces XAP é muita rica e possui vários aspectos além do NoSQL e escalabilidade dados em áreas como gerenciamento de configuração, desenvolvimento, Web Services, integração com produtos de terceiros, etc... A documentação está disponível em http://www.gigaspaces.com/wiki/display/XAP71/Programmer%27s+Guide
Operações NoSQL pode ser implementadas com as seguintes APIs:

- SQLQuery – permite a realização de consulta like-SQL e o uso de expressões regulares; não confundir com JDBC
- Persistência – suporte a vários RDMBS mas também pode implementar outros mecanismos de persistência através dos componentes da API de origem de dados externos.
- Memória cache – suporte a dicionários de par chave/valor disponível para qualquer cliente da rede. As entidades estão disponíveis em todos os nós da rede. A API memória cache é implementada no topo da rede e intercambiada com todas as demais implementações da API.
- Execução de tarefas – permite a execução síncrona e assíncrona de tarefas em nós específicos ou nos clusters.

A API GigaSpaces XAP esta em uma minoria de stemas NoSQL. A maioria dos sistemas NoSQL esforça-se para alcançar a escalabilidade e consistência de dados.

Casos de uso

- Análise em tempo real – análise de dados dinâmicos e relatórios baseados em dados inseridos no sistema em menos de um minuto após seu uso.
- Map/Reduce – processamento de grandes arquivos sobre o sistema de rede
- Tempo de inatividade próximo de zero – permite a alteração dos esquemas dos banco de dados sem o aplicativo mestre/escravo próprio ou proprietário do banco de dados.

Desvantagens

- Complexidade – o servidor, as transações e o modelo de rede são mais complexos que os demais sistemas NoSQL.
- Modelo do servidor de aplicações – A API e os componentes são voltados para a construção de aplicações e lógicas transacionais.
- Curva de aprendizado acentuada – Alto TCO - necessita de uma equipe especializada e bem treinada, mais que os outros sistemas NoSQL.

Mecanismo Google App Datastore

O Datastore é a principal funcionalidade dos aplicativos do Google APP. Ele não é um banco de dados relacionais ou um disfarçado. A API é uma forma pública de acesso ao sistema distribuído de alta performance BigTable da Google. Pense como em uma matriz dispersa distribuída em vários servidores, que permite um número infinito de linhas e colunas. As aplicações podem até definir novas colunas em tempo de execução. O Datastore escala, adicionando novos servidores ao cluster; Google disponibiliza isto sem a participação do usuário.

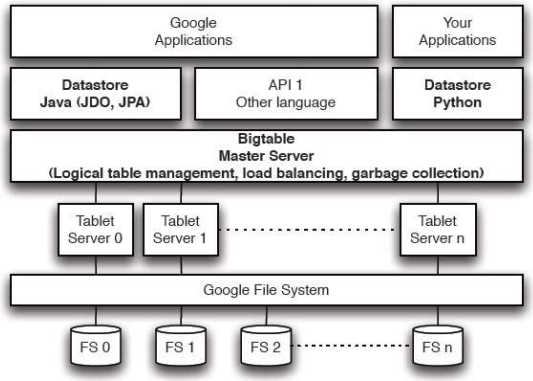

Figura 6 – Arquitetura Datastore

As operações do Datastore são definidas em torno das entidades. As entidades podem ter relacionamentos de um-para-um ou muitos-para-muitos. O Datastore utiliza sua própria chave única ou permite que a aplicação defina a sua. Ele também desabilita algumas propriedades para seu uso próprio. A documentação está disponível em: http://code.google.com/appengine/docs/python/datastore/

Dica quente

Você percebe as semelhanças entre Datastore e MongoDB ? Muitos banco de dados NoSQL resolvem os problemas de forma semelhante

Transações e grupos de entidades

Datastore tem suporte para transações. As transações são efetivadas na entidade que pertencentes ao um grupo. Entidades tem a garantia de serem armazenadas em um mesmo servidor.

Programação Datastore

O modelo de programação Datastore é baseado na herança das entidades básicas, db.Model e db.Expando. A persistência dos dados é mapeada junto a uma especialização da entidade de qualquer uma destas entidades. A API disponibiliza métodos de consulta e persistência para qualquer entidade gerenciada pelo Datastore.

Exemplo programação

A API Datastore é mais simples que as demais e é altamente otimizada para trabalhar no ambiente App Engine. No exemplo abaixo vamos inserir dados e executar uma consulta:


```
from google.appengine.ext import db

class Person(db.Model):
  name = db.StringProperty(required=True)
  age  = db.IntegerProperty(require=True)

person = Person(name = 'Tom', age = 42)

person.put()

person = Person(name = 'Sue', age = 69)

person.put()

# find a specific person

query = Person.all() # every entity!

query.filter('age > ', 20)
query.order('name')

peopleList = query.fetch() # up to 1000

for person in peopleList:
  print person.name
```

O exemplo executas as seguinte operações:

- Define um tipo Pessoa e associa ao Datastore
- Persiste novos itens usando o método put()
- Define e executa uma consulta: informa que o resultado da query será uma string

Gql – Google Query Language

Datastore também permite queries como no formato SQL. A query abaixo mostra como:

```
SELECT * FROM Person WHERE age > 20
    ORDER BY name ASC
```

Gql é mais poderoso para escrever queries do que as escritas nas APIs Python ou Java.

Dica quente

Cuidado!!! As queries escritas em Python e Gql possuem diferenças em performance e execução que podem trazer impactos na execução. Verifique as discussões na documentação Datastore.

Casos de uso

- Escalabilidade massiva – Datastore oferece ótima escalabilidade por utilizar a própria infraestrutura da Google para armazenamento
- Google App Engine – não existe mecanismo alternativo para armazenamento nesta plataforma
- Serviços Web Services RESTful – use Datastore e a infraestrutura App Engine para descarregar tradicionais data centers não nativos e os Web Services necessitam de alta demanda de dados.

Desvantagens

- Forte aderência ao fornecedor – persistência e queries estão fortemente acopladas com a o Datastore e sua API e que estão distantes de serem um padrão da indústria.
- Disponibilidade – Datastore é reconhecido por suas falhas e EULA não permite mais do que 4 em 9 SLAs.
- Custo – o custo de utilização do Datastore é realizado por acesso a dados e tempo de processamento.
- Limites de queries – Result sets possuem um limite de retorno de 1.000 entidades no máximo, forçando as queries serem desnecessariamente complexas.

Permaneça informado

Você quer saber sobre projetos específicos e usos de casos em que NoSQL e escalabilidade de dados são os tópicos quentes? Participe do boletim de escalabilidade:
http://eugeneciurana.com/scalablesystems

Sobre o autor:

Eugene Ciurana é um evangelista do código aberto que se especializou no design e implementação de grandes sistemas de alta disponibilidade e missões críticas. Nos últimos dois anos desenhou e construí sistemas escaláveis em nuvens híbridas para empresas financeiras, software, seguros, e companhias de saúde nos EUA, Japão e Europa. Como principal interlocutor entre a global Walmart.com e a ISD Technology Council, e em 2006 liderou a adoção do Linux e outras tecnologias de código aberto na divisão de sistemas de informação da Walmart.

Publicações:

- Desenvolvendo com Google App Engine
- DZone Refcard #43: escalabilidade e alta disponibilidade
- DZone Refcard #38: padrões SOA
- O testamento Tesla: um suspense

Link para acesso a este documento:
https://dzone.com/refcardz/getting-started-nosql-and-data?oid=hom25105