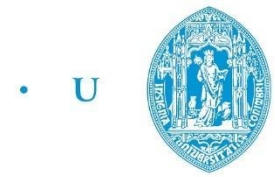
2022/23



FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Trabalho 1

INTEGRAÇÃO DE SISTEMAS

João António Correia Vaz – 2019218159
joaovaz@student.dei.uc.pt
Gonçalo Tavares Antão Folhas Ferreira – 2019214765
tferreira@student.dei.uc.pt

Índice

1. Introdução

No âmbito da cadeira de Integração de Sistemas, o primeiro trabalho tinha como principal objetivo proporcionar-nos as primeiras interações com os formatos XML e *Protocol Buffers.* Assim, foi-nos pedido para analisar e comparar os formatos mencionados tendo em conta os tamanho e velocidade de codificação.

2. Descrição dos Formatos

Como foi mencionado no capítulo anterior, para a realização do Trabalho 1 usámos dois tipos de formatos: XML e *Protocol Buffers*.

2.1. XML

XML é uma ferramenta independente de *software* e *hardware* para armazenar e transportar dados. É uma linguagem de marcação semelhante ao HTML e autoexplicativa. Os dados são organizados de forma hierárquica para que possam ser lidos por vários sistemas e pelos seus utilizadores.

2.2. Protocol Buffers

Os Google *Protocol Buffers* são um mecanismo extensível para serialização das estruturas de dados. São uma combinação entre a linguagem de definição (criada nos ficheiros do tipo proto), o código que o compilador proto gera, as bibliotecas específicas e o formato de serialização para dados que são escritos num ficheiro binário.

3. Código

Criámos algumas estruturas de dados e nesta secção do relatório iremos apresentá-las e fazer uma breve explicação do seu funcionamento.

3.1. Estruturas

3.1.1. Student

A classe *Student* representa o aluno no nosso código, tem sete elementos (nome, telemóvel, género, dia de aniversário, dia de registo, morada e o professor associado) e um atributo: id. No caso desta classe, para o professor associado, guardamos o nome do professor no formato *String*, pois, se tentássemos guardar o objeto inteiro, ao escrever para o XML, estaríamos a criar um ciclo infinito na relação entre *Teacher* e *Student* (fig. 1).

3.1.2. Students

Nesta classe, a única coisa que existe é uma *ArrayList* de *Student*. Deste modo, cada objeto do tipo *Teacher* pode guardar os vários objetos *Student*, isto é, um (objeto) *Teacher* guarda um (objeto) *Students* que, por sua vez, tem todos os (objetos) *Student* que lhe estão associados (fig. 2).

3.1.3. Teacher

À semelhança da classe *Student*, a classe *Teacher* representa o professor no nosso código. Esta tem cinco elementos (nome, data de aniversário, telemóvel, morada e a lista dos seus alunos) e um atributo: id. Para o *Teacher* guardar os alunos, este tem uma variável do tipo *Students*, que por sua vez tem uma *ArrayList* de *Student* (explicado em 3.2) (fig. 3).

3.1.4. Teachers

Por fim, a classe *Teachers* é o objeto que guarda toda a informação, isto é, a lista de todos os professores (que, por sua vez, guardam a lista dos seus alunos) e depois o *marshal* do objeto para XML ou escrito para ficheiro binário com o Google *Protocol Buffers* (fig. 4).

```java
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Student {

    @XmlElement(name = "name")
    String name;
    @XmlElement(name = "phone")
    long phone;
    @XmlElement(name = "gender")
    String gender;
    @XmlElement(name = "birthDate")
    LocalDate birthDate;
    @XmlElement(name = "registrationDate")
    LocalDate registrationDate;
    @XmlElement(name = "address")
    String address;
    @XmlElement(name = "professor")
    String professor;


    @XmlAttribute()
    int id;
```

Figura 1 - Estrutura Student

```java
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Teacher {

    @XmlElement(name = "name")
    String name;
    @XmlElement(name = "birthdate")
    LocalDate birthDate;
    @XmlElement(name = "phone")
    long phone;
    @XmlElement(name = "address")
    String address;
    @XmlElement(name = "students")
    Students students;

    @XmlAttribute()
    int id;
```

Figura 2 – Estrutura Teacher

```java
@XmlRootElement (name = "class")
@XmlAccessorType(XmlAccessType.FIELD)
public class Students {

    @XmlElement(name = "student")
    ArrayList<Student> students;

    public Students() { }

    public void setStudents(ArrayList<Student> students) {
        this.students = students;
    }

    public void addStudent(Student student){
      this.students.add(student);
    }

}
```

Figura 3 - Estrutura Students

```java
@XmlRootElement(name = "class")
@XmlAccessorType(XmlAccessType.FIELD)
public class Teachers {

    @XmlElement(name = "teacher")
    ArrayList<Teacher> list = new ArrayList<>();

    public ArrayList<Teacher> getTeacher() {
      return list;
    }

    public void setTeacher(ArrayList<Teacher> list) {
      this.list = list;
    }

}
```

Figura 4 – Estrutura Teachers

3.2. Funcionamento do código

Nesta secção, explicamos como foi feita a *main* do nosso código. Começamos por inicializar as estruturas e criar um pequeno "banco de dados" para a criação dos alunos e dos professores. De seguida, definimos número de professores e o número de alunos que cada professor irá ter associado. Após esta definição, o código entra num ciclo *for* onde se procede à criação de todos os objetos (para XML e para *Protocol Buffers*), ou seja, dentro deste ciclo *for* existe ainda outro ciclo *for* para realizar a criação dos alunos para um determinado professor.

Quando tudo estiver criado, entramos na parte das contagens de tempo para o XML, XML + GZIP e *Protocol Buffers*. Para a contagem do tempo foi utilizado o método System.nanoTime(). Para o XML, apenas contabilizamos a operação de *marshal*. Já para o XML + GZIP, contabilizamos o tempo que o algoritmo de compressão GZIP demora, acrescido ao tempo de *marshalling* para XML (feito anteriormente). Por fim, para o Google *Protocol Buffers* contamos o tempo de escrita do objeto final (objeto *Teachers*) para um ficheiro binário.

Assim, depois de termos feito a parte de serialização, aplicamos o processo inverso, isto é, a desserialização dos ficheiros. Realizámos a desserialização pela mesma ordem que a serialização: XML primeiro, XML + GZIP em segundo e por fim a descodificação dos *Protocol Buffers*. Tal como na medição de tempo do XML na serialização, também aqui apenas contabilizámos a operação de *unmarshal*. De modo semelhante, o processo de descompressão do GZIP é contado de forma separada e depois acrescentado ao *unmarshal* do XML, calculado antes. Para terminar, é contabilizado o tempo necessário para a realização do parse do ficheiro binário para a obtenção do objeto que fora colocado lá anteriormente.

4. Condições de teste

Para a realização dos testes utilizámos 3 computadores com especificações bastante distintas entre si com o intuito de obter resultados que nos permitam chegar a uma boa conclusão. São eles:

	Computador 1	**Computador 2**	**Computador 3**
Versão Java	JDK 18.0.2.1	JDK 18.0.1	JDK 17.0.2
IDE	Visual Studio Code 1.71.2	Visual Studio Code 1.71.2	Visual Studio Code 1.71.2
Modelo	Torre Asus ROG	Asus ROG GL702VM	ASUS VivoBook 15
RAM	32 GB	16GB	8GB
Processador	Intel(R) Core(TM) i7-8700 CPU	Intel(R) Core(TM) i7-7700HQ CPU	AMD Ryzen 5 3500U

5. Testes realizados

De modo a garantir a fiabilidade e evitar possíveis anomalias na recolha dos dados, para cada par de valores utilizados (número de professores e número de alunos por professor, mencionado em 3.2) foram feitos três testes. Com as três execuções pudemos fazer o cálculo da média e, por sua vez, realizar a representação gráfica dos nossos resultados. As tabelas que deram origem aos gráficos a seguir apresentados encontram-se no capítulo 8.

De relembrar que, como fora dito no ponto 3.2, no início é definido os valores a serem utilizados para a testagem, sendo estes dois parâmetros completamente controlados pelos utilizadores. Nos testes realizados mantivemos o número de alunos por professor fixo (100) e variámos o número de professores (que por sua vez também faz variar o número total de alunos).

Legenda dos gráficos:

Gráficos de serialização/desserialização:

1. XML → serialização/desserialização com XML (em segundos)
2. XML + GZIP → serialização/desserialização com XML com GZIP (em segundo)
3. Protobuf → serialização/desserialização com *Protocol Buffers* (em segundos)

Nota: Os valores obtidos nos testes foram calculados em Bytes, mas para a realização dos gráficos (e armazenamento nas tabelas) foi feita a conversão para MegaBytes. Esta foi feita com base 10 e não com base 2.

5.1. Tabela tempo e Tabela de tamanho (Computador 1)

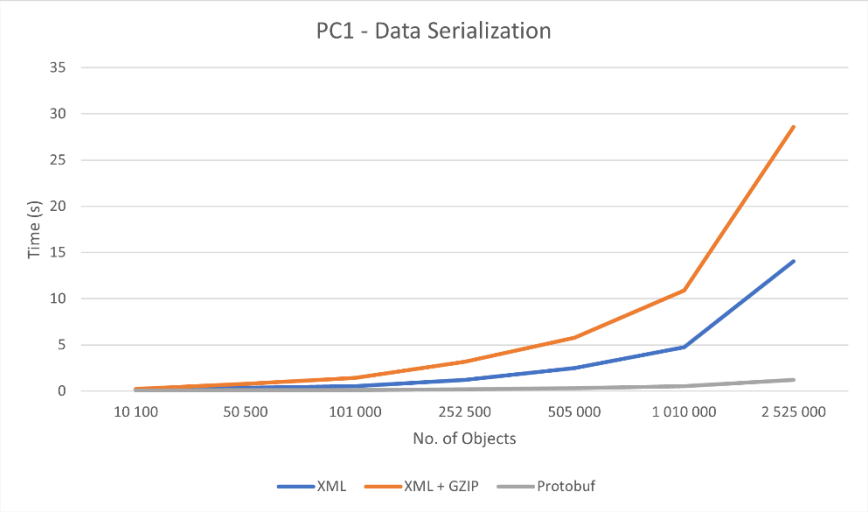


Figura 5 - Gráfico de comparação do processo de Serialização

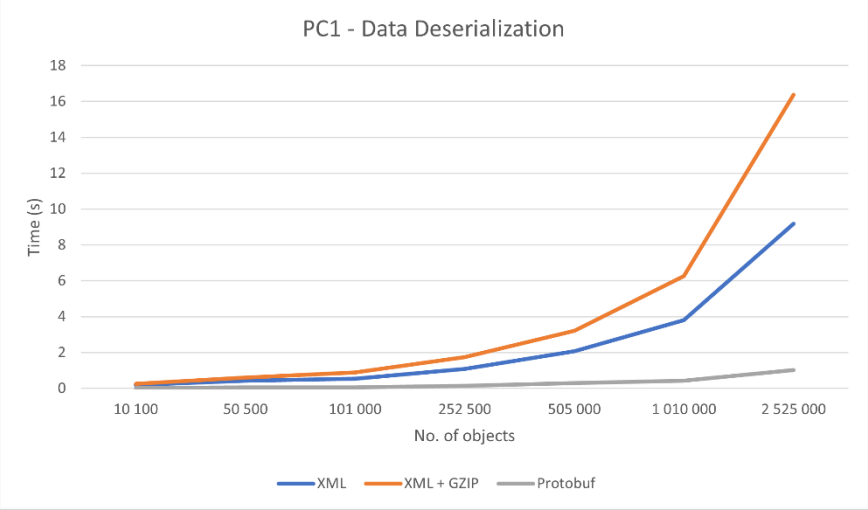


Figura 6 - Gráfico de comparação do processo de Desserialização

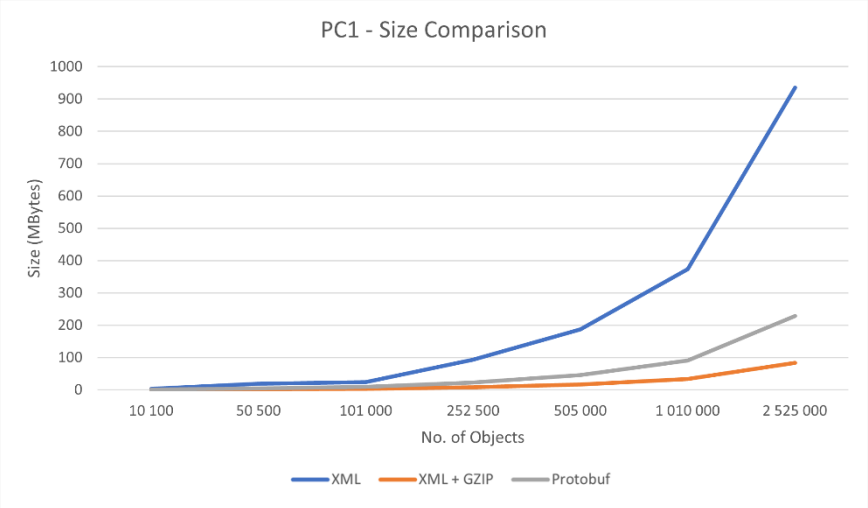


Figura 7 - Gráfico de comparação de tamanho

5.2. Tabela tempo e Tabela de tamanho (Computador 2)

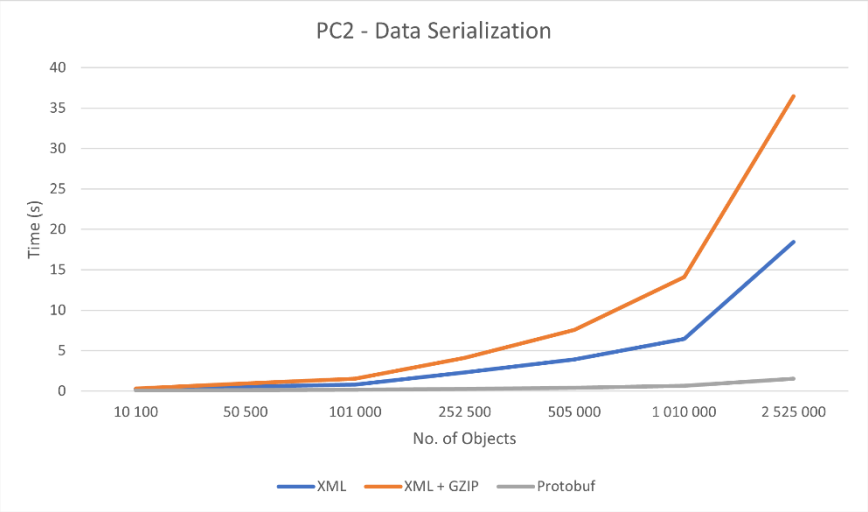


Figura 8 - Gráfico de comparação do processo de Serialização

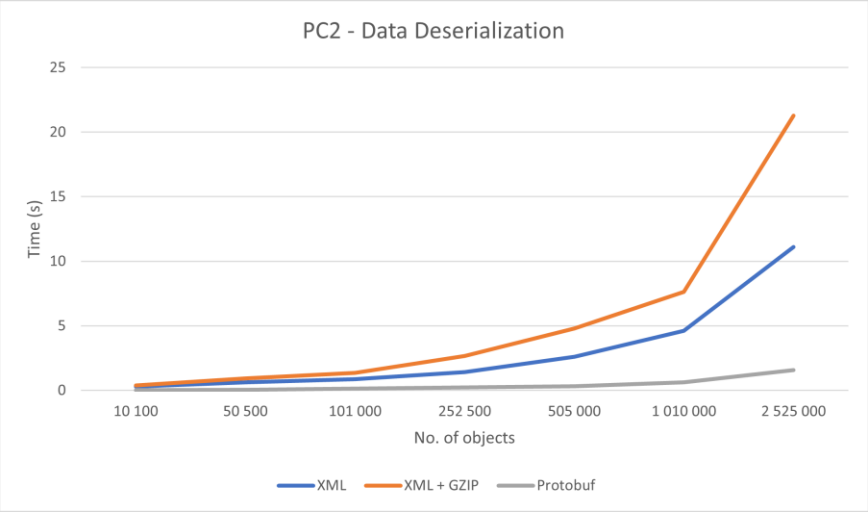


Figura 9 - Gráfico de comparação do processo de Desserialização

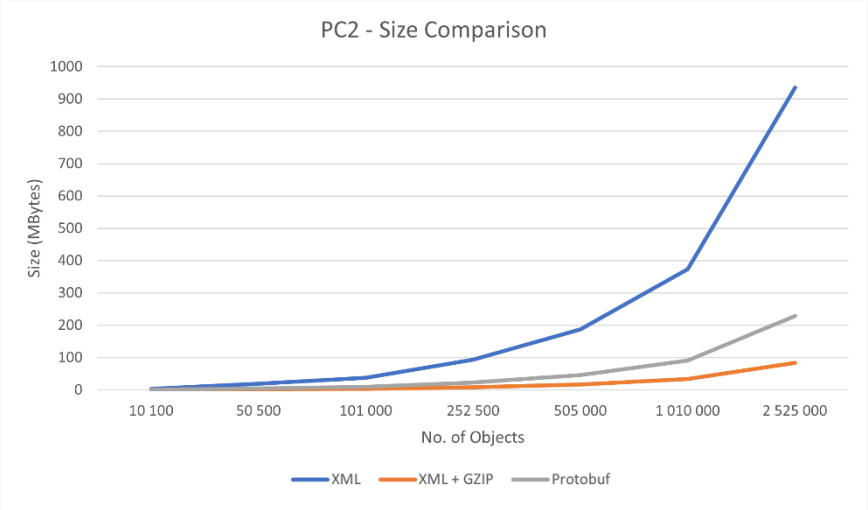


Figura 10 - Gráfico de comparação de tamanho

5.3. Tabela tempo e Tabela de tamanho (Computador 3)

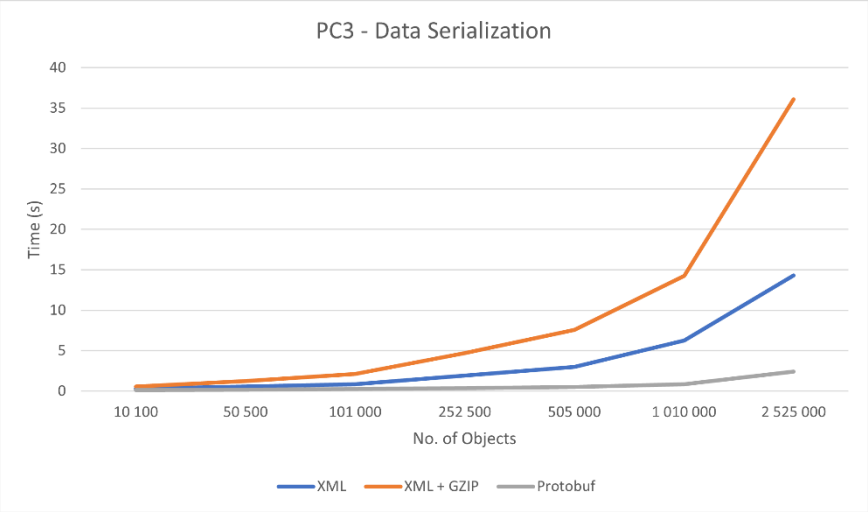


Figura 11 - Gráfico de comparação do processo de Serialização

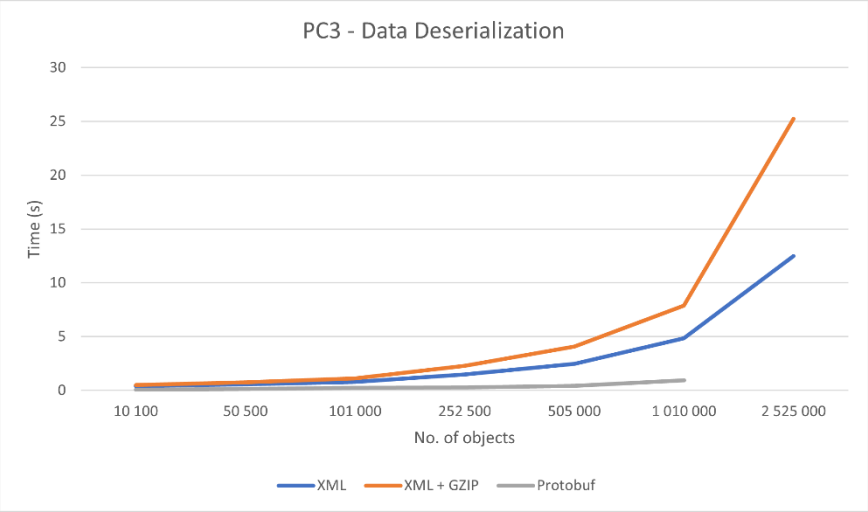


Figura 12 - Gráfico de comparação do processo de Desserialização

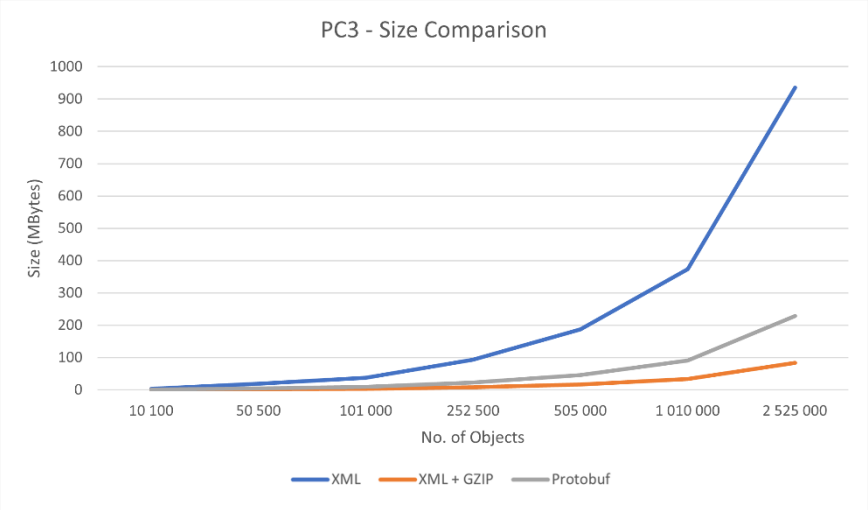


Figura 13 - Gráfico de comparação de tamanho

6. Análise dos resultados

Realizando uma análise rápida aos gráficos apresentados acima, verificamos que os *Protocol Buffers* apresentam melhores resultados a nível de tempo em relação ao XML e ao XML + GZIP. Em contrapartida, no que toca ao tamanho obtido para os ficheiros, o melhor foi o XML + GZIP. Este resultado é expectável tendo em conta que o GZIP é um algoritmo feito para comprimir ficheiros.

Uma explicação possível para os dados obtidos nas tabelas de tempo é que o XML é um formato *human-readable*, pelo que necessita de conter muita mais informação suplementar para ajudar nessa leitura por parte do utilizador. Por outro lado, os *Protocol Buffers* demonstram maior rapidez na serialização/desserialização por gerar um ficheiro binário e por desperdiçar menos recursos.

Para os tamanhos dos ficheiros obtidos na testagem, podemos tirar a conclusão de que o XML, pelos mesmos motivos apresentados no parágrafo anterior, tem um tamanho superior aos restantes por causa da informação toda que tem de armazenar. Em seguida, os *Protocol Buffers* revelaram bons resultados (principalmente em conjunto com os resultados de tempo) no armazenamento da informação criada. Contudo, para os casos maiores (no nosso caso, para valores que gerem mais de 2 milhões de objetos), os *Protocol Buffers* podem dar problemas de utilização de memória. Isto acontece porque, ao trabalhar com grandes quantidades de dados, o utilizador pode acabar com imensas cópias de dados devido às cópias serializadas que são criadas. Por fim, o XML + GZIP apresentou os melhores resultados, tendo sido obtida uma taxa de compressão de 11:1.

Em suma, achamos que, num contexto mais geral, os *Protocol Buffers* são uma melhor opção quando comparado com XML. Porém, apresentam ainda algumas desvantagens para determinadas situações.

7. Webgrafia

1. «Protocol Buffer Basics: Java | Protocol Buffers». Google Developers, https://developers.google.com/protocol-buffers/docs/javatutorial. Acedido 28 de Setembro de 2022.

2. https://www.baeldung.com/google-protocol-buffer. Acedido 20 de Setembro de 2022

3. Tutorial: Protocol Buffer Basics. https://clojusc.github.io/protobuf/current/1050-tutorial.html. Acedido 29 de Setembro de 2022.

4. Protobuf Tutorial. https://www.tutorialspoint.com/protobuf/index.htm. Acedido 27 de Setembro de 2022.

5. Java GZIP Example - Compress and Decompress File | DigitalOcean. https://www.digitalocean.com/community/tutorials/java-gzip-example-compress-decompress-file. Acedido 24 de Setembro de 2022.

6. «Java - How to do Gzip compression of java object». Stack Overflow, 9 de Fevereiro de 2020, https://stackoverflow.com/q/60134395. Acedido 25 de Setembro de 2022.

8. Anexos

Neste capítulo encontram-se as tabelas que deram origem aos gráficos apresentados no capítulo 5 e que apresentam todas as testagens executadas para este relatório. As secções 8.1 e 8.2 dizem respeito aos gráficos do computador 1, 8.3 e 8.4 referem-se ao computador 2 e, por fim, 8.5 e 8.6 pertencem ao computador 3.

Legenda das tabelas:

Tabela de tempo:

1. NTobjs → número total de objetos
2. sXML → serialização com XML (em segundos)
3. sXML + GZIP → serialização com XML com GZIP (em segundo)
4. sProtobuf → serialização com *Protocol Buffers* (em segundos)
5. dXML → desserialização com XML (em segundos)
6. dXML + GZIP → desserialização com XML com GZIP (em segundos)
7. dProtobuf → desserialização com *Protocol Buffers* (em segundos)
8. objCreation → tempo de criação dos objetos para Protocol Buffers (em segundos)

Tabela de tamanho:

1. XML → tamanho do ficheiro XML (em MB)
2. XML + GZIP → tamanho do ficheiro XML quando aplicado o GZIP (em MB)
3. ProtoBuff → tamanho do ficheiro binário gerado pelo *Protocol Buffer* (em MB)

Nota: Os valores obtidos nos testes foram calculados em Bytes, mas para a realização dos gráficos (e armazenamento nas tabelas) foi feita a conversão para MegaBytes. Esta foi feita com base 10 e não com base 2.

MB – MegaBytes

8.1. Tabela de Tempo (Computador 1)

NTobjs	sXML	sXML + GZIP	sProtobuf	dXML	dXML + GZIP	dProtobuf	objCreation
10 100	0,1463	0,2426	0,0424	0,1984	0,2503	0,0196	0,0251
10 100	0,1523	0,2468	0,0420	0,1685	0,2251	0,0168	0,0264
10 100	0,1516	0,2552	0,0459	0,2074	0,2626	0,0197	0,0252
50 500	0,3397	0,7715	0,0867	0,4173	0,5996	0,0554	0,0466
50 500	0,3344	0,7702	0,0876	0,3835	0,5666	0,0453	0,0463
50 500	0,3531	0,7935	0,0878	0,4544	0,6570	0,0525	0,0480
101 000	0,5455	1,4145	0,1143	0,5163	0,8115	0,0573	0,0672
101 000	0,5649	1,4310	0,1178	0,5395	0,9115	0,0613	0,0645
101 000	0,5312	1,4036	0,1268	0,5591	0,9350	0,0622	0,0720
252 500	1,2513	3,1913	0,1768	1,0583	1,7107	0,1216	0,1034
252 500	1,2032	3,1336	0,1984	1,0656	1,6916	0,1320	0,1121
252 500	1,1885	3,1689	0,2075	1,1289	1,7942	0,1410	0,1211
505 000	2,5134	5,7765	0,3311	2,0135	3,1601	0,2692	0.1572
505 000	2,3741	5,5930	0,3036	2,0168	3,1345	0,3037	0,1749
505 000	2,5378	5,9449	0,3454	2,1865	3,3734	0,3304	0,1690
1 010 000	4,5470	10,5600	0,5339	3,6952	6,0433	0,3786	0,3331
1 010 000	4,9380	11,1370	0,5321	3,8498	6,0979	0,5341	0,2667
1 010 000	4,8177	11,0229	0,5359	3,8581	6,6147	0,3873	0,3392
2 525 000	15,3796	29,9724	1,2788	9,3876	16,4484	1,0684	0,6280
2 525 000	15,2547	29,6919	1,2027	9,1754	17,0232	1,0086	0,6856
2 525 000	11,4494	26,1439	1,2007	8,9823	15,6527	1,0037	0,6296

8.2. Tabela de Tamanho (Computador 1)

NTobjs	XML	XML + GZIP	Protobuf
10 100	3,737767	0,335503	0,911985
10 100	3,741185	0,334917	0,915403
10 100	3,734353	0,334937	0,908601
50 500	18,702403	1,673492	4,573793
50 500	18,706024	1,673135	4,577414
50 500	18,704799	1,673736	4,576189
101 000	17,403200	3,345499	9,146090
101 000	17,402111	3,345570	9,145001
101 000	37,406745	3,345104	9,149635
252 500	93,519579	8,366185	22,875469
252 500	93,513736	8,364532	22,869626
252 500	93,506387	8,365963	22,862277
505 000	187,039535	16,731348	45,750425
505 000	187,001020	16,725184	45,711910
505 000	187,014594	16,725726	45,725726
1 010 000	374,041375	33,454928	91,462265
1 010 000	374,017785	33,452652	91,438675
1 010 000	374,065834	33,451079	91,486724
2 525 000	935,053465	83,649520	228,597971
2 525 000	935,197535	83,650883	228,742041
2 525 000	935,108342	83,655129	228,652848

8.3. Tabela de Tempo (computador 2

NTobjs	sXML	sXML + GZIP	sProtobuf	dXML	dXML + GZIP	dProtobuf	objCreation
10 100	0,2508	0,3249	0,0589	0,2957	0,3666	0,0332	0,0413
10 100	0,2577	0,3306	0,0617	0,2799	0,3548	0,0229	0,0440
10 100	0,2762	0,3591	0,0577	0,3030	0,3812	0,0355	0,0439
50 500	0,5411	0,9408	0,1084	0,5348	0,8235	0,0640	0,0704
50 500	0,5186	0,8974	0,1687	0,7051	0,9530	0,0540	0,1147
50 500	0,5943	0,9378	0,1101	0,6281	0,9913	0,0485	0,0743
101 000	0,8055	1,5004	0,2374	1,0550	1,5140	0,1625	0,1094
101 000	0,7469	1,4623	0,1559	0,7859	1,3000	0,1099	0,1260
101 000	0,8977	1,5862	0,1594	0,7407	1,2715	0,1167	0,0859
252 500	2,6241	4,3521	0,2512	1,4249	2,6569	0,2604	0,1404
252 500	2,2992	4,0655	0,2685	1,4276	2,7466	0,1932	0,1804
252 500	2,0017	3,8931	0,3237	1,3598	2,5764	0,2032	0,1751
505 000	4,2017	7,5348	0,4293	2,5127	4,8155	0,3251	0,2228
505 000	3,9268	7,4870	0,4272	2,6941	5,2095	0,3128	0,2447
505 000	3,6316	7,7422	0,3853	2,5755	4,3878	0,3635	0,2017
1 010 000	6,4360	14,0404	0,6482	4,5965	7,2776	0,5245	0,3917
1 010 000	5,8747	13,5890	0,6439	4,5936	8,0013	0,7287	0,4047
1 010 000	7,0019	14,6225	0,6350	4,6718	7,6247	0,6113	0,4123
2 525 000	17,9591	36,1319	1,5045	11,1619	20,2676	1,6158	0,8168
2 525 000	20,2356	38,3347	1,5714	11,1679	22,7028	1,5554	0,8526
2 525 000	17,1389	34,9377	1,5076	11,0083	20,8872	1,5325	0,9003

8.4. Tabela de Tamanho (Computador 2)

NTobjs	XML	XML + GZIP	Protobuf
10 100	3,738517	0,335281	0,912735
10 100	3,744176	0,335095	0,918394
10 100	3,739085	0,334885	0,913303
50 500	18,697042	1,673836	4,568432
50 500	18,707522	1,673002	4,578912
50 500	18,692893	1,673033	4,564283
101 000	37,404558	3,346521	9,147448
101 000	37,408602	3,345860	9,151492
101 000	37,411268	3,344662	9,145158
252 500	93,507638	8,365152	22,863528
252 500	93,497560	8,364857	22,853450
252 500	93,509702	8,365943	22,865592
505 000	186,988862	16,730968	45,699752
505 000	187,014251	16,724463	45,725141
505 000	187,024633	16,728744	45,735523
1 010 000	374,062820	33,457110	91,483710
1 010 000	374,051709	33,453924	91,472599
1 010 000	374,003601	33,454653	91,424491
2 525 000	935,116155	83,650723	228,660661
2 525 000	935,057406	83,648247	228,601912
2 525 000	935,168016	83,655566	228,712522

8.5. Tabela de Tempo (Computador 3)

NTobjs	sXML	sXML + GZIP	sProtobuf	dXML	dXML + GZIP	dProtobuf	objCreation
10 100	0,3292	0,5726	0,1174	0,4475	0,5953	0,0888	0,0440
10 100	0,2954	0,4998	0,1374	0,3956	0,4794	0,0525	0,0618
10 100	0,3499	0,5655	0,1410	0,3444	0,4617	0,0741	0,0544
50 500	0,5815	1,2141	0,1682	0,5105	0,6993	0,1249	0,0921
50 500	0,5942	1,2772	0,1766	0,5693	0,8073	0,1336	0,0869
50 500	0,5236	1,1614	0,1805	0,6027	0,7891	0,1272	0,0914
101 000	0,8617	2,1983	0,2692	0,8000	1,1808	0,2998	0,1426
101 000	0,8994	2,1330	0,2320	0,7521	1,0812	0,1675	0,1149
101 000	0,8504	2,0772	0,2282	0,7723	1,0980	0,2134	0,1604
252 500	1,8060	4,5991	0,3802	1,4161	2,1945	0,3356	0,2183
252 500	2,1278	4,9691	0,3802	1,6075	2,3531	0,2623	0,2324
252 500	1,8440	4,5603	0,3440	1,4465	2,2565	0,2651	0,2018
505 000	2,8123	7,1725	0,5346	2,5021	4,1864	0,3968	0,3520
505 000	3,0317	7,6304	0,5249	2,3748	3,9694	0,4617	0,2993
505 000	3,0936	7,8595	0,5165	2,4778	4,1013	0,4419	0,2644
1 010 000	5,7757	13,8408	0,8432	4,8758	7,8026	0,8791	0,4773
1 010 000	7,3500	15,4478	0,8663	4,9487	8,1990	0,8971	0,4412
1 010 000	5,5760	13,4803	0,8480	4,7252	7,6661	1,0436	0,4902
2 525 000	13,5446	34,3664	2,5355	12,1362	24,3924	*error*	0,9397
2 525 000	14,7612	36,9864	2,7360	12,3181	25,1135	*error*	1,0273
2 525 000	14,6634	36,8291	1,9480	12,9604	26,2212	*error*	0,9571

8.6. Tabela de Tamanho (Computador 3)

NTobjs	XML	XML + GZIP	Protobuf
10 100	3,744670	0,334993	0,918888
10 100	3,737640	0,334983	0,911858
10 100	3,738270	0,335341	0,912488
50 500	18,702612	1,673700	4,574002
50 500	18,703044	1,673753	4,574434
50 500	18,701115	1,672845	4,572505
101 000	37,398133	3,345830	9,141023
101 000	37,398501	3,345599	9,141391
101 000	37,414147	3,344140	9,157037
252 500	93,517638	8,366000	22,873528
252 500	93,500082	8,365466	22,855972
252 500	93,505373	8,365485	22,861263
505 000	187,019909	16,726060	45,730799
505 000	187,038832	16,725978	45,749722
505 000	187,017324	16,728875	45,728214
1 010 000	374,054495	33,454863	91,475385
1 010 000	374,023961	33,458434	91,444851
1 010 000	374,039580	33,453909	91,460470
2 525 000	935,089668	83,649022	228,634174
2 525 000	935,104625	83,649870	228,649131
2 525 000	935,087745	83,648849	228,632251